Mach Learn
DOI 10.1007/s10994-016-5558-8

Probabilistic logic programming for hybrid relational domains

Davide Nitti[1] · **Tinne De Laet**[2] · **Luc De Raedt**[1]

Received: 14 December 2014 / Accepted: 23 February 2016

Abstract We introduce a probabilistic language and an efficient inference algorithm based on distributional clauses for static and dynamic inference in hybrid relational domains. Static inference is based on sampling, where the samples represent (partial) worlds (with discrete and continuous variables). Furthermore, we use backward reasoning to determine which facts should be included in the partial worlds. For filtering in dynamic models we combine the static inference algorithm with particle filters and guarantee that the previous partial samples can be safely forgotten, a condition that does not hold in most logical filtering frameworks. Experiments show that the proposed framework can outperform classic sampling methods for static and dynamic inference and that it is promising for robotics and vision applications. In addition, it provides the correct results in domains in which most probabilistic programming languages fail.

Keywords Probabilistic programming · Statistical relational learning · Discrete and continuous distributions · Particle filter · Likelihood weighting · Logic programming

Davide Nitti is supported by the IWT (Agentschap voor Innovatie door Wetenschap en Technologie).

Editor: Filip Železný.

✉ Davide Nitti
davide.nitti@cs.kuleuven.be

Tinne De Laet
Tinne.DeLaet@kuleuven.be

Luc De Raedt
luc.deraedt@cs.kuleuven.be

[1] Department of Computer Science, KU Leuven, Leuven, Belgium

[2] Faculty of Engineering Science, KU Leuven, Leuven, Belgium

Published online: 22 April 2016

1 Introduction

Robotics research has made important achievements in problems such as state estimation, planning, and learning. However, the majority of probabilistic models used, such as Bayesian networks, cannot easily represent relational information, that is, objects, properties as well as the relations that hold between them. Relational representations allow one to encode more general models, to integrate background knowledge about the world, and to convert low-level information into human-readable form. Probabilistic programming languages (De Raedt et al. 2008) and statistical relational learning techniques (SRL) (Getoor and Taskar 2007) provide such relational representations and have been successful in many application areas ranging from natural language processing to bioinformatics.

This paper extends probabilistic logic programming techniques to deal with hybrid relational domains, involving both discrete and continuous random variables in two settings. The first setting is a static one, where we contribute a new inference algorithm for Distributional Clauses (DC) (Gutmann et al. 2011), a recent extension of Sato's distribution semantics (Sato 1995) for dealing with continuous variables. The second setting is a dynamic one, where we extend the DC framework for coping with time. For the resulting Dynamic Distributional Clauses (DDC), we develop a particle filter (DCPF) that exploits the static inference algorithm for filtering. Particle filters (Doucet et al. 2000) are widely applied in domains such as probabilistic robotics (Thrun et al. 2005), and we adapt them here for use in hybrid relational domains, in which each state of the environment is represented as an interpretation, that is, a set of ground facts that defines a possible world. The statistical relational learning literature already contains several approaches to temporal models and to particle filters; see Sect. 8 for a detailed discussion. However, few frameworks are suited for tracking or other robotics applications: they are too slow for online applications or they only support discrete domains.

The distinguishing features of the proposed framework are that: (1) it provides an inference method that extends the applicability of likelihood weighting (LW) and works with zero probability evidence; (2) it exploits partial worlds as samples allowing for a potentially infinite state space; (3) it employs a relational representation to represent (context-specific) independence assumptions, and exploits these to speed up inference; (4) it is suited for tracking and robotics applications; and (5) it has a bounded space complexity for filtering by avoiding inference backward in time (backinstantiation).

The contributions of this paper are that (1) we propose an improved sampling algorithm for static inference; (2) we extend DC for dynamic domains (DDC); (3) we introduce an optimized particle filter for DDC; (4) we prove the theoretical correctness for DCPF and study its relation with Rao–Blackwellized particle filters; (5) we integrate online learning in the DCPF; and (6) we adopt the DCPF in some tracking scenarios. This paper is based on previous papers (Nitti et al. 2013, 2014c) (plus a position paper Nitti et al. 2014a and a summary paper Nitti et al. 2014b), but is extended with contributions (1) and (4), and additional experiments.

This paper is organized as follows: we first review DC (Sect. 2), introduce some extensions and present the static inference procedure (Sect. 3). We then introduce the Dynamic DC, the particle filters (Sect. 4), and propose the DCPF (Sect. 5). We then integrate learning in DCPF (Sect. 6), present some experiments (Sect. 7), discuss related work (Sect. 8), and conclude (Sect. 9).

2 Distributional clauses

We now introduce the distributional clauses (DC) (Gutmann et al. 2011), an extension of distribution semantics (Sato 1995), while assuming some familiarity with statistical relational

learning and logic programming (De Raedt et al. 2008) (see "Logic programming" in the Appendix for a brief overview).

Formally, a *distributional clause* is a formula of the form $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n$, where the $\mathtt{b}_i$ are literals and $\sim$ is a binary predicate written in infix notation.
The intended meaning of a distributional clause is that each ground instance of the clause $(\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n)\theta$ defines the random variable $\mathtt{h}\theta$ with distribution $\mathcal{D}\theta$ whenever all the $\mathtt{b}_i\theta$ hold, where θ is a substitution. In other words, a distributional clause can be seen as a powerful template to define conditional probabilities: $p(\mathtt{h}\theta|(\mathtt{b}_1, \ldots, \mathtt{b}_n)\theta) = \mathcal{D}\theta$. The term $\mathcal{D}$ can be nonground, i.e., values, probabilities, or distribution parameters can be related to conditions in the body. Furthermore, given a random variable r, the term $\simeq(r)$ constructed from the reserved functor $\simeq/1$ represents the value of r. Abusing notation, for brevity, we shall sometimes write $r \sim\!\!= v$ instead of $\simeq(r) = v$, which is true iff the value of the random variable r unifies with v.

Example 1 Consider the following clauses:

$$\mathtt{n} \sim \mathtt{poisson}(6). \tag{1}$$

$$\mathtt{pos(P)} \sim \mathtt{uniform(0,M)} \leftarrow \mathtt{n} \sim\!\!= \mathtt{N}, \mathtt{between(1,N,P)}, \mathtt{M\ is\ 10 * N}. \tag{2}$$

$$\mathtt{left(A,B)} \leftarrow \simeq\!\mathtt{(pos(A))} < \simeq\!\mathtt{(pos(B))}. \tag{3}$$

where '$\mathtt{is}$' is the equality operator. Capitalized terms such as $\mathtt{P}$, $\mathtt{A}$, and $\mathtt{B}$ are logical variables, which can be substituted with any constant. Clause (1) states that the number of people $\mathtt{n}$ is governed by a Poisson distribution with mean 6. Clause (2) models the position $\mathtt{pos(P)}$ as a continuous random variable uniformly distributed from 0 to $\mathtt{M} = 10\mathtt{N}$ (that is, 10 times the number of people), for each person identifier $\mathtt{P}$ such that $1 \leq \mathtt{P} \leq \mathtt{N}$, where $\mathtt{P}$ is integer and $\mathtt{N}$ is unified with the number of people $\mathtt{n}$. For example, if the value of $\mathtt{n}$ is 2, there will be 2 independent random variables $\mathtt{pos(1)}$ and $\mathtt{pos(2)}$ with distribution $\mathtt{uniform(0,20)}$. Finally, clause (3) defines the binary relation $\mathtt{left}$, comparing people positions. Note that the atom $\mathtt{left(a,b)}$ is defined using a deterministic clause, but it is a random variable as it depends on other random variables.

DCs support continuous distributions (under reasonable conditions) and naturally cope with an unknown number of objects (Gutmann et al. 2011). In addition to distributional clauses, we shall also employ deterministic clauses as in Prolog. We shall often talk about clauses when the context is clear.

A distributional program $\mathbb{P}$ is a set of distributional and/or deterministic clauses that defines a distribution $p(x)$ over possible worlds x. The probability $p(q)$ of a query q can be estimated using Monte-Carlo methods, that is, possible worlds are sampled from $p(x)$, and $p(q)$ is approximated as the ratio of samples in which the query q is true.

The procedure used to generate possible worlds defines the semantics and a basic inference algorithm. A possible world is generated starting from the empty partial world $x = \emptyset$; then for each distributional clause $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n$, whenever the body $\{\mathtt{b}_1\theta, \ldots, \mathtt{b}_n\theta\}$ is true in the set x for the substitution θ, a value v for the random variable $\mathtt{h}\theta$ is sampled from the distribution $\mathcal{D}\theta$ and $\mathtt{h}\theta = v$ is added to the new partial world $\hat{x}$. This is also performed for deterministic clauses, adding ground atoms to $\hat{x}$ whenever the body is true. This process is then recursively repeated until a fixpoint is reached, that is, until no more variables can be sampled and added to the world. The final world is called complete or full, while the intermediate worlds are called partial. The process is based on sampling, thus 'world' is often replaced with sample or particle. Notice that a possible world may contain a countably infinite number of random variables (and atoms).

Example 2 Given the DC program $\mathbb{P}$ defined by (1), (2), and (3), $\{\mathtt{n} = 2, \mathtt{pos}(1) = 3.1, \mathtt{pos}(2) = 4.5, \mathtt{left}(1, 2)\}$ is a possible (complete) world. This world is sampled in the following order: $\emptyset \rightarrow \{\mathtt{n} = 2\} \rightarrow \{\mathtt{n} = 2, \mathtt{pos}(1) = 3.1, \mathtt{pos}(2) = 4.5\} \rightarrow \{\mathtt{n} = 2, \mathtt{pos}(1) = 3.1, \mathtt{pos}(2) = 4.5, \mathtt{left}(1, 2)\}$.

Gutmann et al. (2011) formally describe this generative process using the ST_P operator. To define a proper probability distribution $p(x)$, $\mathbb{P}$ needs to satisfy the validity conditions described in Gutmann et al. (2011). See "Distributional clauses" in the Appendix for details on the validity conditions and the ST_P operator. Throughout the paper the partial worlds will be written as x^P, while $x \supseteq x^P$ indicates a complete world consistent with x^P, and $x^a = x \setminus x^P$ represents the remaining part of the world.

We extended DC to allow for negated literals in the body of distributional clauses. To accommodate negation, we need to consider the case that a random variable is not defined in a full world x. Any comparison involving a non-defined variable will fail; therefore, its negation will succeed. In contrast, grounded atoms a are considered false in x, if $a \notin x$ (standard closed-world assumption). For further details on negation in DC see "Negation" in the Appendix.

3 Static inference for distributional clauses

Sampling full worlds is generally inefficient or may not even terminate as possible worlds can be infinitely large. Therefore, Gutmann et al. (2011) use magic sets (Bancilhon and Ramakrishnan 1986) to generate only those facts that are relevant for answering the query. Magic sets are a well-known logic programming technique for forward reasoning. In this paper, we propose a more efficient sampling algorithm based on backward reasoning and likelihood weighting.

3.1 Importance sampling

Given a program $\mathbb{P}$, the probability of the query q is estimated by applying importance sampling to partial samples $x^{P(i)}$, with $i = 1, \ldots, N$ where N is the number of samples. In importance sampling the proposal probability $g(x)$ used to generate samples is not necessary the target probability $p(x)$.

The Monte-Carlo approximation is the following (where $\mathbb{1}$ is the indicator function[1]):

$$
\begin{aligned}
p(q) &= E_{p(x)}[\mathbb{1}(x \models q)] = \int_x \mathbb{1}(x \models q)\, p(x)\, dx = \int_{x^P} \int_{x^a} \mathbb{1}(x \models q)\, p(x^a|x^P)\, p(x^P)\, dx^a\, dx^P \\
&= \int_{x^P} \underbrace{\int_{x^a} \mathbb{1}(x \models q)\, p(x^a|x^P)\, dx^a}_{p(q|x^P)}\, p(x^P)\, dx^P = \int_{x^P} p(q|x^P)\, p(x^P)\, dx^P \\
&= \int_{x^P} \underbrace{p(q|x^P)\frac{p(x^P)}{g(x^P)}}_{w_q}\, g(x^P)\, dx^P \approx \frac{1}{N}\sum_{i=1}^{N} w_q^{(i)}, \qquad (4)
\end{aligned}
$$

[1] The indicator function is 1 when the argument is true, zero otherwise.

where the weight is $w_q^{(i)} = p(q|x^{P(i)})\frac{p(x^{P(i)})}{g(x^{P(i)})}$. Formula (4) uses a fixed split $x = x^P \cup x^a$. Following Milch (2006) we extend this idea to exploit context-specific independencies: we can have a different split in different samples.

There are two reasons to sample partial worlds instead of complete ones. First, the sampling process is faster and terminates (under some conditions) even when the complete world is a countably infinite set. Second, the estimator variance is generally lower with respect to a naive Monte-Carlo estimator that samples complete worlds. Indeed, sampling some variables $x^{P(i)}$ and computing $p(q|x^{P(i)})$ analytically is an instance of the conditional Monte-Carlo method (Lemieux 2009) (sometimes called Rao–Blackwellization), which has better performance with respect to naive Monte-Carlo.

The split of x has to guarantee that the probability $p(q|x^{P(i)})$ is analytically computable. Let $var(q)$ denote the set of all random variables in q. If $var(q) \subseteq var(x^{P(i)})$, all the variables in q are instantiated in $x^{P(i)}$, thus q can be determined deterministically: $p(q|x^{P(i)}) = \mathbb{1}(x^{P(i)} \models q)$. In some cases it is possible to compute $p(q|x^{P(i)})$ without sampling variables in q.

Example 3 Given the clauses (1), (2), (3),

$$p(q|x^{P(i)}) = p(\texttt{pos(1)} > 5 \mid \{\texttt{n} = 3, \texttt{pos(1)} = 3\}) = 0,$$

while the marginalized variables `pos(2)`, `pos(3)`, `left(X, Y)` are irrelevant. We could even do better, for example $p(q|x^{P(i)}) = p(\texttt{pos(1)} > 5|\{\texttt{n} = 3\})$ is analytically computable without sampling $var(q) = \{\texttt{pos(1)}\}$.

In general, it is impossible to sample only $var(q)$ as the DC program does not directly define the distribution of these variables. For example, to sample `pos(1)` defined by (2) we first need to sample `n` defined by (1); thus we need to follow the generative sampling process (ST_P operator) until the variables of interest are sampled. Backward reasoning (or the magic set transformation) can help to focus the sampling.

The probability of a query given evidence $p(q|e)$ can be estimated using formula (4) twice, once to estimate $p(q, e)$ and another time to estimate $p(e)$, then $p(q|e)$ is approximated as the ratio of the two quantities. As an alternative, formula (4) can be adapted to use the same partial samples to estimate the two quantities. In detail, each sample is split in $x = x_e^P \cup x_q^P \cup x^a$, such that x_e^P is sufficient to compute $p(e|x_e^P)$ and $x_q^P \cup x_e^P$ is sufficient to compute $p(q|e, x_q^P, x_e^P)$. Formula (4) is applied to estimate $p(e)$, then the same partial samples $x_e^{P(i)}$ are expanded to estimate $p(q, e)$:

$$\begin{aligned}
p(q,e) &= \int_{x_e^P}\int_{x_q^P} p(q,e|x_q^P,x_e^P)p(x_q^P,x_e^P)dx_q^P dx_e^P \\
&= \int_{x_e^P}\int_{x_q^P} p(q|e,x_q^P,x_e^P)p(e|x_e^P)p(x_q^P|x_e^P)p(x_e^P)dx_q^P dx_e^P \\
&= \int_{x_e^P}\int_{x_q^P} p(q|e,x_q^P,x_e^P)p(x_q^P|x_e^P)dx_q^P\, p(e|x_e^P)p(x_e^P)dx_e^P \\
&= \int_{x_e^P}\int_{x_q^P} p(q|e,x_q^P,x_e^P)\frac{p(x_q^P|x_e^P)}{g(x_q^P|x_e^P)}g(x_q^P|x_e^P)dx_q^P\, p(e|x_e^P)\frac{p(x_e^P)}{g(x_e^P)}g(x_e^P)dx_e^P
\end{aligned}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} \underbrace{p(q|e, x_q^{P(i)}, x_e^{P(i)})\frac{p(x_q^{P(i)}|x_e^{P(i)})}{g(x_q^{P(i)}|x_e^{P(i)})}}_{w_q^{(i)}} \underbrace{p(e|x_e^{P(i)})\frac{p(x_e^{P(i)})}{g(x_e^{P(i)})}}_{w_e^{(i)}}$$

$$= \frac{1}{N}\sum_{i=1}^{N} w_q^{(i)} w_e^{(i)}$$

$$p(q|e) = \frac{p(q,e)}{p(e)} \approx \frac{\sum_{i=1}^{N} w_q^{(i)} w_e^{(i)}}{\sum_{i=1}^{N} w_e^{(i)}}, \tag{5}$$

where the proposal g has the same factorization of the target distribution: $g(x_q^P, x_e^P) = g(x_q^P|x_e^P)g(x_e^P)$.

3.2 Sampling partial possible worlds

We now present our approach to sampling possible worlds and computing $p(q)$ following Equation (4) and $p(q|e)$ following (5). Central is the algorithm with signature

EvalSampleQuery(q : query, $x^{P(i)}$: partial world) **returns** $(w_q^{(i)}, x_q^{P(i)})$

that starts from a given query q and a partial world $x^{P(i)}$ (which will be empty in case there is no evidence, cf. below), and generates an expanded partial world $x_q^{P(i)}$ together with its weight $w_q^{(i)}$ so that

1. $x_q^{P(i)} \supseteq x^{P(i)}$, i.e., $x_q^{P(i)}$ is an expansion of $x^{P(i)}$ obtained using ST_P,
2. $var(q) \subseteq var(x_q^{P(i)})$, which ensures that we can evaluate q in $x_q^{P(i)}$ and therefore $p(q|x_q^{P(i)})$,
3. $w_q^{(i)} = p(q|x_q^{P(i)})\frac{p(x_q^{P(i)}|x^{P(i)})}{g(x_q^{P(i)}|x^{P(i)})}$.

$p(q)$ is estimated by calling $(w_q^{(i)}, x_q^{P(i)}) \leftarrow$ EvalSampleQuery$(q, \emptyset)$ N times and applying (4). The probability $p(q|e)$ is estimated by calling, for each sample, $(w_e^{(i)}, x_e^{P(i)}) \leftarrow$ EvalSampleQuery$(e, \emptyset)$ for the evidence, and $(w_q^{(i)}, x_q^{P(i)}) \leftarrow$ EvalSampleQuery$(q, x_e^{P(i)})$ for the query given $x_e^{P(i)}$, and then applying (5).

The key question is thus how to sample one such partial world $x_q^{P(i)}$ for a generic call of EvalSampleQuery$(q, x^{P(i)})$. To realize this, we combine likelihood weighting (LW) (Fung and Chang 1989; Koller and Friedman 2009) with a variant of SLD-resolution in the EvalSampleQuery algorithm that we describe below.

SLD-resolution (Apt 1997; Lloyd 1987; Nilsson and Małiszyński 1995) is an inference procedure to prove a query q, used in logic programming, that focuses the proof on the relevant part of the program $\mathbb{P}$. The basic idea is replacing an atom with its definition. Given $q = (q_1, q_2, \ldots, q_n)$ and a rule $head \leftarrow body \in \mathbb{P}$ such that $\theta = mgu(q_1, head)$ (i.e. $q_1\theta = head\theta$), then the new goal becomes $q' = (body, q_2, \ldots, q_n)\theta$, where mgu indicates the most general unifier. If the empty goal is reached then the query is proved. If it is impossible to reach the empty goal, the query is assumed false under the closed-world assumption. There may be more than one rule that satisfies the mentioned conditions, resulting in the SLD-tree. In Prolog the tree is traversed using depth-first search with backtracking.

Likelihood weighting is a type of importance sampling that forces variables to be consistent with the evidence by using an adapted proposal distribution g. It has been shown (Fung and Chang 1989) that LW reduces the variance of the estimator with respect to the naive Monte-Carlo estimator. In this paper LW is also used to force variables to be consistent with the query. LW has connections with conditional Monte Carlo (for binary events). Indeed, computing $p(r = v|x^{P(i)})$ is equivalent to imposing $r = v$ and weight the sample with $w = p(r = v|x^{P(i)})$.

Adapting SLD-resolution. EVALSAMPLEQUERY employs an extension of SLD-resolution to determine which random variables to sample, until the query q can be evaluated. However, unlike traditional SLD-resolution, it keeps track of a number of global variables:

1. the weight $w_q^{(i)}$, initialized to 1,
2. the initial query iq, initialized to q, and
3. the partial sample $x^{P(i)}$.

Starting from a goal $G = q$, EVALSAMPLEQUERY applies inference rules until the goal G is empty (i.e., q has been proven) or no more rules can be applied (q fails). If the query succeeds, the algorithm returns the final weight and the expanded sample $(w_q^{(i)}, x_q^{P(i)})$. If the query fails it returns $(w_q^{(i)} = 0, x_q^{P(i)})$.

Given a goal G and the global variables $w_q^{(i)}, iq, x^{P(i)}$, applying a rule produces a new goal G' and modifies the global variables:

1. G' is the new goal obtained from G using a kind of SLD-resolution step;
2. if a new variable r is sampled with value v,
   - set $w_q^{(i)} \leftarrow w_q^{(i)} \frac{p(r=v|x^{P(i)})}{g(r=v|x^{P(i)})}$ (based on LW) and
   - $x^{P(i)} \leftarrow x^{P(i)} \cup \{r = v\}$.

   In addition, if $r \sim\!= Val \in iq$ and $(r \sim\!= v, iq) \Leftrightarrow iq\theta$ with r grounded and $\theta = \{Val = v\}$ then:
   - $iq \leftarrow iq\theta$
3. if a new atom h is proved, set $x^{P(i)} \leftarrow x^{P(i)} \cup \{h\}$.

The inference rules applied by EVALSAMPLEQUERY resemble SLD resolution applied to a query q: they are applied with a backtracking strategy, negation as failure to prove negated literals (as in SLDNF) and tabling to improve performance. However, important differences are required to handle the stochastic nature of sampling and to exploit LW whenever possible. Note that the partial sample $x^{P(i)}$ can only grow during the application of the inference rules, i.e., backtracking does not remove sampled values from the partial sample. This is necessary to guarantee the generation of a sample from the defined proposal distribution g.

The algorithm needs the original query to apply LW. Since the inference rules change the current goal to prove, we distinguish the current goal G from the original query $iq = q$; during sampling iq can be simplified (e.g., applying substitutions) as long as $\mathbb{P} \models (x^{P(i)}, iq) \Leftrightarrow (x^{P(i)}, q)$, i.e. iq is logically equivalent to q given $x^{P(i)}$ and the DC program $\mathbb{P}$. For those reasons $x^{P(i)}$, $w_q^{(i)}$, and iq are global variables.

Let us assume that the query $q = (q_1, q_2, \ldots, q_n)$ contains only equality comparisons for random variables, e.g., $r \sim\!= v$. Any other comparison operator $r \odot v$ can be converted in $r \sim\!= Val, Val \odot v$, where Val is a logical variable and $Val \odot v$ is ground during evaluation. For example, $\simeq(n) > 5$ becomes $n \sim\!= N, N > 5$. The inference rules are the following:

1a. If $[\exists h \in x^{P(i)} : \theta = mgu(q_1, h)]$ OR $[builtin(q_1), \exists \theta : q_1\theta]$ then:

$$(q_1, q_2, \ldots, q_n) \vdash (q_2, \ldots, q_n)\theta$$

i.e., if $q_1\theta$ is true in $x^{P(i)}$ for a substitution θ, remove q_1 from the current goal and apply the substitution θ to the current goal. q_1 can also be a built-in predicate such as $1 < 4$ that is trivially proved.

1b. If $\exists \theta$ s.t. $h \leftarrow body \in \mathbb{P}, \theta = mgu(q_1, h)$ then:

$$(q_1, q_2, \ldots, q_n) \vdash (body, add(h), q_2, \ldots, q_n)\theta$$

i.e., if q_1 unifies with the head of a deterministic clause, then add the body of the clause and $add(h)$ to the current goal, and apply substitution θ. The special predicate $add(h)$ indicates that h must be added to $x^{P(i)}$ after the body has been proven.

2a. If $\exists \theta$ s.t. $h = v \in x^{P(i)}, \theta = mgu(q_1, h \sim= v)$:

$$(q_1, q_2, \ldots, q_n) \vdash (q_2, \ldots, q_n)\theta$$

i.e., if $q_1\theta$ compares a sampled random variable h to a value and $q_1\theta$ is true in $x^{P(i)}$, then remove q_1 from the current goal and apply substitution θ.

2b. If $\exists \theta$ s.t. $h \sim \mathcal{D} \leftarrow body \in \mathbb{P}, \theta = mgu(q_1, h \sim= Val), h \notin var(x^{P(i)})$:

$$(q_1, q_2, \ldots, q_n) \vdash (body, sample(h, \mathcal{D}), q_1, q_2, \ldots, q_n)\theta$$

i.e., if q_1 compares a (not yet sampled) variable h that unifies with the head of a DC clause, then add the body of the clause and $sample(h, \mathcal{D})$ to the current goal and apply the substitution θ; $sample(h, \mathcal{D})$ is a special predicate that indicates that we need to sample h from $\mathcal{D}$ and add $h = val$ to $x^{P(i)}$ after the body has been proven.

3a. If $(h \sim= v) \in iq, ground(h \sim= v), h \notin var(x^{P(i)}), [(h \neq v, x^{P(i)}) \models \neg iq]$:

$$\begin{aligned}(sample(h, \mathcal{D}), q_2, \ldots, q_n) \vdash\ & (q_2, \ldots, q_n) \\ & w_q^{(i)} \leftarrow w_q^{(i)} \cdot likelihood_{\mathcal{D}}(h = v) \\ & x^{P(i)} \leftarrow x^{P(i)} \cup \{h = v\}\end{aligned}$$

i.e., if $sample(h, \mathcal{D})$ is in the current goal, $h \sim= v$ is ground in iq, and $h \neq v$ makes iq false (always true if iq is a conjunction of literals), and h is not sampled in $x^{P(i)}$, then add $h = v$ to $x^{P(i)}$, weight accordingly (LW), and remove $sample(h, \mathcal{D})$ from the current goal.

3b. If $h \notin var(x^{P(i)})$, $ground(h)$, and rule 3a is not applicable:

$$\begin{aligned}(sample(h, \mathcal{D}), q_2, \ldots, q_n) \vdash\ & (q_2, \ldots, q_n) \\ & x^{P(i)} \leftarrow x^{P(i)} \cup \{h = v\}\end{aligned}$$

$$\text{if } h \sim= Val \in iq, ((h \sim= v, iq) \Leftrightarrow iq\gamma) \text{ then } iq \leftarrow iq\gamma$$

with v sampled from $\mathcal{D}$, and $\gamma = \{Val = v\}$. That is, if $sample(h, \mathcal{D})$ is in the current goal, and rule 3a is not applicable, then sample h, add it to $x^{P(i)}$, and remove $sample(h, \mathcal{D})$ from the current goal. Finally, apply the substitution γ to iq iff $iq\gamma$ is equivalent to iq with $h \sim= v$ (always true if iq is a conjunction of literals).

3c. If $ground(h)$:

$$\begin{aligned}(add(h), q_2, \ldots, q_n) \vdash\ & (q_2, \ldots, q_n) \\ & x^{P(i)} \leftarrow x^{P(i)} \cup \{h\}\end{aligned}$$

i.e., if $add(h)$ is in the current goal and h is ground, then add h to $x^{P(i)}$ and remove $add(h)$ from the current goal.

EVALSAMPLEQUERY performs lazy instantiation exploiting context-specific independencies: only the random variables needed to answer the query are sampled, the values of the remaining random variables are irrelevant to determine the true value of q for that specific partial instantiation.

Theorem 1 *For $N \to \infty$ samples generated using* EVALSAMPLEQUERY*, the estimation $\hat{p}(q)$ obtained using (4) converges with probability 1 to the correct probability $p(q)$.*

Proof It is sufficient to prove that EVALSAMPLEQUERY satisfies the importance sampling requirement for which convergence guarantees are available (Robert and Casella 2004), that is $\forall x : p(q|x)p(x) > 0 \Rightarrow g(x) > 0$ or equivalently: $\forall x : g(x) = 0 \Rightarrow p(q|x)p(x) = 0$. The algorithm samples random variables h using the target distribution when LW is not applied (rule 3b): $g(h|x^{P(i)}) = p(h|x^{P(i)})$. LW is applied with proposal $g(h = val|x^{P(i)}) = 1$ for grounded equalities in the initial query $(h \sim= val) \in iq$ (rule 3a). Therefore, $g(h \neq val, x^{P(i)}) = 0$ but also $p(q|h \neq val, x^{P(i)})p(h \neq val, x^{P(i)}) = 0$, because the query q fails for $h \neq val$. Indeed, $(h \neq val, x^{P(i)}) \models \neg iq$ as required in rule 3a, and it is easy to show that $\mathbb{P} \models (x^{P(i)}, iq) \Leftrightarrow (x^{P(i)}, q)$, therefore $(\mathbb{P}, h \neq val, x^{P(i)}) \models \neg q$. The requirement is thus satisfied $(\forall x : g(x) = 0 \Rightarrow p(q|x)p(x) = 0)$. □

Theorem 1 is extendable for conditional probabilities $p(q|e) = p(q, e)/p(e)$, as long as $p(e) > 0$. The remainder of this section will consider $p(e) = 0$; it can be safely skipped by the reader less interested in technical details.

Zero probability evidence. Special considerations need to be made for queries with zero probability evidence. For example, when the evidence is $h \sim= val$ with h a continuous random variable defined with a density distribution. Such conditional distributions are not unambiguously defined, and a reformulation of the problem, e.g., a change of variables, can produce a different result (Borel–Kolmogorov paradox (Kadane 2011, Chap. 5.10). To avoid these issues we need to make some assumptions. In this section we make an explicit distinction between probabilities P, and densities p: $\frac{d}{dx}P(X \leq x) = p(x)$. Following Kadane (2011), we define the conditional probability for zero probability evidence as follows:

$$\begin{aligned} P(q|e = v) &= \lim_{dv \to 0} \frac{P(q, e \in [v - dv/2, v + dv/2])}{P(e \in [v - dv/2, v + dv/2])} \\ &= \lim_{dv \to 0} \frac{\int_x \mathbb{1}(x \models q)p(x, e = v)dxdv}{p(e = v)dv} \\ &= \frac{\int_x \mathbb{1}(x \models q)p(x, e = v)dx}{p(e = v)}, \end{aligned} \tag{6}$$

The conditional density is thus $p(x|e = v) = \frac{p(x,e=v)}{p(e=v)}$. If the evidence is a disjunction, the limit might not exists, e.g.,:

$$P(q|e = a \vee e = b) = \lim_{\substack{da \to 0 \\ db \to 0}} \frac{P(q, (e \in [a - da/2, a + da/2] \vee e \in [b - db/2, b + db/2]))}{P(e \in [a - da/2, a + da/2] \vee e \in [b - db/2, b + db/2])}$$

$$= \lim_{\substack{da \to 0 \\ db \to 0}} \frac{P(q, e \in [a - da/2, a + da/2]) + P(q, e \in [b - db/2, b + db/2]))}{P(e \in [a - da/2, a + da/2]) + P(e \in [b - db/2, b + db/2])}$$

$$= \lim_{\substack{da\to 0\\ db\to 0}} \frac{\int_x \mathbb{1}(x \models q)p(x, e = a)dxda + \int_x \mathbb{1}(x \models q)p(x, e = b)dxdb}{p(e = a)da + p(e = b)db}.$$

To avoid this issue, we assume $da = db$. In this case we obtain

$$P(q|e = a \vee e = b) = \frac{\int_x \mathbb{1}(x \models q)p(x, e = a)dx + \int_x \mathbb{1}(x \models q)p(x, e = b)dx}{p(e = a) + p(e = b)}.$$

For example, the probability of nationality given a height of $180cm$ or $160cm$ is defined as $P(nationality|height \in [180 - da/2, 180 + da/2]\ OR\ height \in [160 - db/2, 160 + db/2])$ for $da \to 0, db \to 0$. The limit depends on how da and db relate to each other. Setting $da = db$ seems a reasonable assumption when the intervals are comparable quantities, but other assumptions are possible. Definition (6) is extendable to more complex distributions that involve multiple variables and mixtures of discrete and continuous variables.

To apply importance sampling to definition (6), it is sufficient to estimate $P(e \in [v - dv/2, v + dv/2])$ and $P(q, e \in [v - dv/2, v + dv/2])$ for $dv \to 0$. Knowing that $dr \to 0 \Rightarrow P(h \in [r - dr/2, r + dr/2]) \to p(h = r)dr$, every time we apply LW to a continuous variable, $h = r$ is intended as $h \in [r - dr/2, r + dr/2]$ with $dr \to 0$, thus the incremental weight in rule 3a is $p(h = r)dr$.

Formula (5) needs the sum of importance weights. This has to be carefully computed when there is a mix of densities and probability masses (Owen 2013, Chap. 9.8). Imagine that there is a sample weight $w_1 = P(x)$ obtained assigning a discrete variable to a value, and a second sample weight $w_2 = p(y)dy$ obtained assigning a continuous variable to a value, or more precisely to a range $[y - dy/2, y + dy/2]$, with $dy \to 0$. The weight w_1 **trumps** w_2, because the latter goes to zero. Indeed, the second sample has a weight infinitely smaller than the first, and thus it is ignored in the weight sum: $w_1 + w_2 = P(x) + p(y)dy = P(x)$ (for $dy \to 0$). Analogously, a weight $w_a = p(x_1, \ldots, x_n)dx_1, \ldots, dx_n$ that is the product of n (one-dimensional) densities trumps a weight $w_b = p(x'_1, \ldots, x'_n, x'_{n+1}, \ldots, x'_{n+m})dx_1, \ldots, dx_{n+m}$ that is the product of n densities of the same variables and $m > 0$ other densities (i.e. $w_a + w_b = w_a$). If all the weights are n-dimensional densities (of the same variables), then the quantities are comparable and are trivially summed. However, if the weights refer to different variables, we need an assumption to ensure the existence of the limit, e.g., $\forall i : dx_i = dx$, thus $dx^{n+m}/dx^n \to 0$, making again n-dimensional densities trump $(n + m)$-dimensional densities. For $m = 0$ the densities are trivially summed, e.g., $w_1 = p(a, b, c)dadbdc$ and $w_2 = p(f, g, e)dfdgde$, assuming $dadbdc = dfdgde = dx^3$, we obtain $w_1 + w_2 = (p(a, b, c) + p(f, g, e))dx^3$. Finally, the ratio of weights sums in (5) is computed assuming again $\forall i : dx_i = dx$, obtaining $P(q|e) \approx \lim_{dx\to 0}(k_n dx^v)/(k_d dx^l)$. If $v > l$ then $P(q|e) = 0$, otherwise for $v = l$ we have $P(q|e) \approx k_n/k_d$. Those distinctions are automatically performed in EvalSampleQuery. For zero probability evidence we do not have convergence results for every DC program, query, and evidence, nonetheless the inference algorithm produces the correct results in many domains, as shown in the next section and in the experiments.

3.3 Examples

We now illustrate EvalSampleQuery and the cases when LW can be applied with the following example.

Example 4

$$\texttt{n} \sim \texttt{uniform}([1,2,3,4,5,6,7,8,9,10]). \tag{7}$$

$$\texttt{color}(\texttt{X}) \sim \texttt{uniform}([\texttt{grey},\texttt{blue},\texttt{black}]) \leftarrow \texttt{material}(\texttt{X}) \sim\!= \texttt{metal}. \tag{8}$$

$$\texttt{color}(\texttt{X}) \sim \texttt{uniform}([\texttt{black},\texttt{brown}]) \leftarrow \texttt{material}(\texttt{X}) \sim\!= \texttt{wood}. \tag{9}$$

$$\texttt{material}(\texttt{X}) \sim \texttt{finite}([0.3:\texttt{wood}, 0.7:\texttt{metal}]) \leftarrow \texttt{n} \sim\!= \texttt{N}, \texttt{between}(1,\texttt{N},\texttt{X}). \tag{10}$$

$$\texttt{drawn}(\texttt{Y}) \sim \texttt{uniform}(\texttt{L}) \leftarrow \texttt{n} \sim\!= \texttt{N}, \texttt{findall}(\texttt{X}, \texttt{between}(1,\texttt{N},\texttt{X}), \texttt{L}). \tag{11}$$

$$\texttt{size}(\texttt{X}) \sim \texttt{beta}(2,3) \leftarrow \texttt{material}(\texttt{X}) \sim\!= \texttt{metal}. \tag{12}$$

$$\texttt{size}(\texttt{X}) \sim \texttt{beta}(4,2) \leftarrow \texttt{material}(\texttt{X}) \sim\!= \texttt{wood}. \tag{13}$$

We have an urn, where the number of balls n is a random variable and each ball X has a color, material, and size with a known distribution. The i-th ball drawn with replacement from the urn is named drawn(i). The special predicate findall(A, B, L) finds all A that makes B true and puts them in a list L. In (11) L is the list of integers from 1 to N, where N is unified with the number of balls.

Let us consider the query p(color(2) ∼= black), the derivation is the following (omitting $iq = q$):

1: $(\texttt{color}(2) \sim\!= \texttt{black}); w_q^{(i)} = 1; x^{P(i)} = \emptyset$
↓2b on (8) :
2: $(\texttt{material}(2) \sim\!= \texttt{metal}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim\!= \texttt{black}); w_q^{(i)} = 1; x^{P(i)} = \emptyset$
↓2b on (10) :
3: $(\texttt{n} \sim\!= \texttt{N}, \texttt{between}(1,\texttt{N},2), \texttt{sample}(\texttt{material}(2), \mathcal{D}_{\texttt{material}(2)}), \texttt{material}(2) \sim\!= \texttt{metal},$
$\texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim\!= \texttt{black}); w_q^{(i)} = 1;\ x^{P(i)} = \emptyset$
↓2b on (7) :
4: $(\texttt{sample}(\texttt{n}, \mathcal{D}_{\texttt{n}}), \texttt{n} \sim\!= \texttt{N}, \texttt{between}(1,\texttt{N},2), \texttt{sample}(\texttt{material}(2), \mathcal{D}_{\texttt{material}(2)}),$
$\texttt{material}(2) \sim\!= \texttt{metal}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim\!= \texttt{black}); w_q^{(i)} = 1; x^{P(i)} = \emptyset$
↓3b :
5: $(\texttt{n} \sim\!= \texttt{N}, \texttt{between}(1,\texttt{N},2), \texttt{sample}(\texttt{material}(2), \mathcal{D}_{\texttt{material}(2)}), \texttt{material}(2) \sim\!= \texttt{metal},$
$\texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim\!= \texttt{black}); w_q^{(i)} = 1;\ x^{P(i)} = \{\texttt{n} = 3\}$
↓2a followed by 1a
6: $(\texttt{sample}(\texttt{material}(2), \mathcal{D}_{\texttt{material}(2)}), \texttt{material}(2) \sim\!= \texttt{metal}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)})$
$\texttt{color}(2) \sim\!= \texttt{black}); w_q^{(i)} = 1;\ x^{P(i)} = \{\texttt{n} = 3\}$
↓3b :
7: $(\texttt{material}(2) \sim\!= \texttt{metal}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim\!= \texttt{black})$
$w_q^{(i)} = 1;\ x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}\}$
8: fail, backtracking to 1

↓2b on (9) :
9: $(\texttt{material}(2) \sim\!= \texttt{wood}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim\!= \texttt{black})$
$w_q^{(i)} = 1;\ x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}\}$
↓2a :
10: $(\texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim\!= \texttt{black}); w = 1; x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}\}$
↓3a :
11: $(\texttt{color}(2) \sim\!= \texttt{black}); w_q^{(i)} = 1/2;\ x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}, \texttt{color}(2) = \texttt{black}\}$
↓1a :
12: $\square; w_q^{(i)} = 1/2;\ x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}, \texttt{color}(2) = \texttt{black}\}$

The algorithm starts checking whether a rule is applicable to the current goal initialized with the query. For example, rule 2a fails because `color(2)` is not in the sample $x^{P(i)}$. Rule 2b can be applied to clause (8), obtaining tuple 2. At this point `material(2)` needs to be evaluated, it is not sampled and it unifies with the head of clause (10), thus applying rule 2b we obtain tuple 3. Now '`n`' is required, thus it is sampled with value e.g, 3 (rules 2b on (7) and 3b), for which '`n` $\sim=$ `N, between(1, N, 2)`' succeeds for `N` $= 3$ (rule 2a and 1a). At tuple 6 the body of (10) has been proven; therefore, `material(2)` is sampled with a value e.g., `wood` (rule 3b). Now in tuple 7, the formula `material(2)` $\sim=$ `metal` fails because the sampled value is `wood` (and thus the body of clause (8) fails); the algorithm backtracks to tuple 1 and applies rule 2b on clause (9) obtaining tuple 9. This time `material(2)` $\sim=$ `wood` is true and can be removed from the current goal (rule 2a). At this point `color(2)` needs to be sampled (tuple 10). LW is applied because `color(2)` is in the original query $iq = q$, thus `color(2)` $=$ `black` is added to the sample with weight $1/2$ (rule 3a). The query in this sample is true with final weight $1/2$.

3.3.1 Query expansion

Instead of asking for `color(2)` $\sim=$ `black`, let us add `a` $\leftarrow$ `color(2)` $\sim=$ `black` to the DC program and ask $p($`a`$)$; the query does not change. However, the described rules do not apply LW because $iq =$ `a`, and `a` is deterministic. It is clear that LW should be applicable also in this case. To solve this issue it is sufficient to expand iq, replacing each literal in iq by its definition; e.g., $iq =$ `a` becomes $iq =$ (`color(2)` $\sim=$ `black`). At this point the algorithm is able to apply LW as before. We can go even further, replacing $iq =$ (`color(2)` $\sim=$ `black`) with $iq =$ (`color(2)` $\sim=$ `black`, (`material(2)` $\sim=$ `metal` `OR` `material(2)` $\sim=$ `wood`)), where the disjunction of the bodies that define `color(2)` has been added. Note that for random variables we need to keep the literal in iq after the expansion (e.g., `color(2)` $\sim=$ `black`). Indeed, the disjunction of the bodies guarantees only that the random variable exists, not that it takes the value black. This procedure is a type of partial evaluation (Lloyd and Shepherdson 1991) adapted for probabilistic DC programs. There are several ways to unfold (expand) a query, one possible way is the following. Before applying the inference rules, the initial query iq is set to the disjunction of all proofs of q without sampling: $iq = (proof_1\ OR\ proof_2\ OR \ldots OR\ proof_n)$. Each proof is determined using SLD-resolution (a number of unfolding operations); since random variables are not sampled, if the truth value of a literal cannot be determined because it is non-ground (e.g., `N` > 0, or `between(A, B, C)`) it is left unchanged in the proof. Starting from $proof = q$, a proof can be found as follows:

- e1 replace each deterministic atom $h \in proof$ with $body\theta$ where $head \leftarrow body \in \mathbb{P}$ and $\theta = mgu(h, head)$, the process is repeated recursively for $body\theta$;
- e2 if the truth value of $h \in proof$ cannot be determined it is left unchanged;
- e3 for each $h \sim= value \in proof$ add $body\theta$ where $head \sim \mathcal{D} \leftarrow body \in \mathbb{P}$ and $\theta = mgu(h, head)$, the process is repeated recursively for $body\theta$.

A depth limit is necessary for recursive clauses. The obtained formula can be simplified, e.g., $(a, b)\ OR\ (a, d)$ becomes $a, (b\ OR\ d)$; this is useful to know which variables can be forced to be true (in the example 'a').

After the iq expansion the sampling algorithm can start using the same inference rules, that cover the case in which iq contains disjunctions. As described in rule 3a, we can apply LW setting $h = val$, only when $(h \neq val, x^{P(i)}) \models \neg iq$. Thus, if $iq =$ ((`h` $\sim=$ `val` `OR` `a`), `b`),

LW cannot be applied for $\texttt{h} \sim= \texttt{val}$. However, if a becomes false, iq simplifies to $(\texttt{h} \sim= \texttt{val}, \texttt{b})$, and LW can be applied setting $h = val$, because $h \neq val$ makes iq false. To determine whether $(h \neq val, x^{P(i)}) \models \neg iq$ holds, it is convenient to simplify iq whenever a random variable is sampled. For example, after a random variable h has been sampled with value v, $\texttt{h} \sim= \texttt{val}$ is replaced with its truth value (true or false). Furthermore, (true OR a) is simplified with true, (false OR a) with a, and so on. The expansion of iq and the simplification guarantee that $\mathbb{P} \models (x^{P(i)}, iq) \Leftrightarrow (x^{P(i)}, q)$ as required by Theorem 1. The iq expansion allows to exploit LW in a broader set of cases. Indeed, it is basically a form of partial evaluation adapted for DCs and being able to exploit similar optimizations, e.g., using constraint propagation where the constraints that make the query true are propagated with the iq expansion, and updated according to the sampled variables.

3.3.2 Complex queries

Example 5 A more complex query is $\simeq(\texttt{color(2)}) = \simeq(\texttt{color(1)})$, which is converted to $\texttt{color(2)} \sim= \texttt{Y}, \texttt{color(1)} \sim= \texttt{Y}$ as in this way each subgoal refers to a single random variable. In this case, color(2) is sampled (rule 3b: LW is not used) for example to red (after sampling n and material). Assuming $\texttt{n} \geq 2$ the first subgoal $\texttt{color(2)} \sim= \texttt{Y}$ succeeds with substitution $\gamma = \{\texttt{Y} = \texttt{red}\}$, thus the original query becomes $iq = (\texttt{color(2)} \sim= \texttt{red}, \texttt{color(1)} \sim= \texttt{red})$. The remaining subgoal will be $\texttt{color(1)} \sim= \texttt{red}$ for which LW is used (rule 3a). Indeed, the original query iq becomes grounded and LW can be applied.

The examples show that EvalSampleQuery exploits LW in complex queries (or evidence). This is also valid for continuous random variables for which MCMC or naive MC will return 0. The probability of such queries is 0, but $p(e) = 0$ is not a satisfactory answer to estimate $p(q|e)$, and the limit (6) needs to be computed. For simple evidence or queries (e.g., $\texttt{size(1)} \sim= \texttt{v}$) classical LW is sufficient to solve the problem. For more complex evidence (e.g., $\simeq(\texttt{size(1)}) = \simeq(\texttt{size(2)})$) MCMC, naive MC or classical LW will fail to provide an answer (all samples are rejected). Many probabilistic languages cannot handle those queries (if we exclude explicit approximations such as discretization). In contrast, the proposed algorithm is able to provide a meaningful answer.

Example 6 Let us consider the Indian GPA problem (Perov et al.) proposed by Stuart Russell. According to Perov et al. "Stuart Russell [...] pointed out that most probabilistic programming systems [...] produce the wrong answer to this problem". The reason for this is that contemporary probabilistic programming languages do not adequately deal with mixtures of density and probability mass distributions. The proposed inference algorithm for DC does provide the correct results for the Indian GPA problem. The DC program that defines the domain is the following:

$$\texttt{isdensityA} \sim \texttt{finite}([0.95 : \texttt{true}, 0.05 : \texttt{false}]).$$
$$\texttt{agpa} \sim \texttt{beta}(8, 2) \leftarrow \texttt{isdensityA} \sim= \texttt{true}.$$
$$\texttt{americanGPA} \sim \texttt{finite}([0.85 : 4.0, 0.15 : 0.0]) \leftarrow \texttt{isdensityA} \sim= \texttt{false}.$$
$$\texttt{americanGPA} \sim \texttt{val(V)} \leftarrow \texttt{agpa} \sim= \texttt{A}, \texttt{V is A} * 4.0.$$
$$\texttt{isdensityI} \sim \texttt{finite}([0.99 : \texttt{true}, 0.01 : \texttt{false}]).$$
$$\texttt{igpa} \sim \texttt{beta}(5, 5) \leftarrow \texttt{isdensityI} \sim= \texttt{true}.$$
$$\texttt{indianGPA} \sim \texttt{finite}([0.1 : 0.0, 0.9 : 10.0]) \leftarrow \texttt{isdensityI} \sim= \texttt{false}.$$

$$\texttt{indianGPA} \sim \texttt{val(V)} \leftarrow \texttt{igpa} \sim\!= \texttt{A}, \texttt{V is A} * 10.0.$$
$$\texttt{nation} \sim \texttt{finite([0.25 : america, 0.75 : india])}.$$
$$\texttt{studentGPA} \sim \texttt{val(A)} \leftarrow \texttt{nation} \sim\!= \texttt{america}, \texttt{americanGPA} \sim\!= \texttt{A}.$$
$$\texttt{studentGPA} \sim \texttt{val(I)} \leftarrow \texttt{nation} \sim\!= \texttt{india}, \texttt{indianGPA} \sim\!= \texttt{I}.$$

where $\texttt{h} \sim \texttt{val(v)}$ means that h has value v with probability 1. Briefly, the student GPA has a mixed distribution that depends on the student nationality. An interesting query is $p(\texttt{nation} \sim\!= \texttt{america} | \texttt{studentGPA})$. For example, for $\texttt{studentGPA} = 4$ such probability is 1. The proposed inference algorithm provides the correct result for this query. This is due to the proper estimation of limit (6) and the relative importance weights. Moreover, the iq expansion and the generalized LW avoid rejection-sampling issues with continuous evidence.

Example 7 The last case to discuss is a query that contains random variables that are nonground terms, e.g., $\texttt{color(X)} \sim\!= \texttt{black}$, which is interpreted as $\exists \texttt{X}\ \texttt{color(X)} \sim\!= \texttt{black}$. In this case LW is not applied because the goal is nonground. Applying LW would produce wrong results because we would force the value black only for the first proof (e.g., $\texttt{color(1)} \sim\!= \texttt{black}$), ignoring the other possible proofs (e.g., $\texttt{color(2)} \sim\!= \texttt{black}$), and thus violating the importance sampling requirement. In some cases, query expansion can enumerate all possible grounded proofs, making LW applicable.

LW can also be applied for the query $\simeq(\texttt{material}(\simeq(\texttt{drawn(1)}))) = \texttt{wood}$ (the first drawn ball is made of wood) which is converted to $(\texttt{drawn(1)} \sim\!= \texttt{X}, \texttt{material(X)} \sim\!= \texttt{wood})$. Once drawn(1) is sampled to a value v (without LW), the substitution $\theta = \{\texttt{X} = \texttt{v}\}$ is applied to the current goal and to iq. At this point LW can be applied to $\texttt{material(v)} \sim\!= \texttt{wood}$ because it is grounded in iq (rule 3a). In other words, for the partial world $x^{P(i)} = \{\texttt{drawn(1)} = \texttt{v}\}$ the only value of material(v) that makes the query true is wood for which LW is applicable. For this query one sample is sufficient to obtain the exact result.

4 Dynamic distributional clauses

We now extend Distributional Clauses to Dynamic Distributional Clauses (DDC) for modeling dynamic domains. We then discuss how inference by means of filtering is realized in the propositional case.

4.1 Dynamic distributional clauses

As in input-output HMMs and in planning, we shall explicitly distinguish between the hidden state x_t, the evidence or observations z_t, and the action u_t (input). Therefore, in DDCs, each predicate/variable is classified as state x, action u, or observation z, with a subscript that refers to time 0, for the initial step; time t for the current step, and $t+1$ for the next step. The definition of a discrete-time stochastic process follows the same idea of a Dynamic Bayesian Network (DBN). We need sets of clauses that define:

1. the prior distribution: $\texttt{h}_0 \sim \mathcal{D} \leftarrow \texttt{body}_0$,
2. the state transition model : $\texttt{h}_{\texttt{t+1}} \sim \mathcal{D} \leftarrow \texttt{body}_{\texttt{t:t+1}}$ (the body involves variables at time t and eventually at time $t+1$),
3. the measurement model: $\texttt{z}_{\texttt{t+1}} \sim \mathcal{D} \leftarrow \texttt{body}_{\texttt{t+1}}$, and

4. clauses that define a random variable at time t from other variables at the same time (intra-time dependence): $\mathtt{h}_\mathtt{t} \sim \mathcal{D} \leftarrow \mathtt{body}_\mathtt{t}$.

Obviously, deterministic clauses are allowed in the definition of the stochastic process. As these clauses are all essentially distributional clauses, the semantics remains unchanged.

Example 8 Let us consider a dynamic model for the position of 2 objects: a ball and a box. For the sake of clarity we consider one-dimensional positions.

$$\mathtt{pos(ID)}_0 \sim \mathtt{uniform}(0, 1) \leftarrow \mathtt{between}(1, 2, \mathtt{ID}). \tag{14}$$

$$\mathtt{pos(ID)}_{\mathtt{t}+1} \sim \mathtt{gaussian}(\simeq(\mathtt{pos(ID)}_\mathtt{t}), 0.1). \tag{15}$$

$$\mathtt{obsPos(ID)}_{\mathtt{t}+1} \sim \mathtt{gaussian}(\simeq(\mathtt{pos(ID)}_{\mathtt{t}+1}), 0.01). \tag{16}$$

The object positions have a uniform distribution at step 0 (14). The next position of each object is equal to the current position plus Gaussian noise (15). In addition, the observation model (16) for each object is a Gaussian distribution centered in the actual object position. As in Example 1, we can also define the number of objects as a random variable.

4.2 Inference and filtering

If we consider the time step as an argument of the random variable, inference can be done as in the static case with no changes. However, the performance will degrade, while time and space complexity will grow linearly with the maximum time step considered in the query. This problem can be mitigated if we are interested in filtering, as we shall show in the next section.

The problem of filtering has been well studied in the propositional case (e.g., for DBNs). It is concerned with the estimation of the current state x_t when the state is only indirectly observable through observations z_t. Formally, filtering is concerned with estimating the belief, that is, the probability density function $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$. The Bayes filter computes recursively the belief at time $t+1$, starting from the belief at t, the last observation z_{t+1}, and the last action performed u_{t+1} through

$$bel(x_{t+1}) = \eta p(z_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}|x_t, u_{t+1}) bel(x_t) dx_t,$$

where η is a normalization constant. Even in the propositional case, the above integral is only tractable for specific combinations of distributions $bel(x_t)$, $p(x_{t+1}|x_t, u_{t+1})$, and $p(z_{t+1}|x_{t+1})$ [e.g., the Kalman filter (Kalman 1960)], and one therefore has to resort to approximations. One solution used in the propositional case is to use Monte-Carlo techniques for approximating the integral, resulting in the particle filter (Doucet et al. 2000). The key idea of particle filtering is to represent the belief by a set of weighted samples (often called particles). Given N weighted samples $\{(x_t^{(i)}, w_t^{(i)})\}$ distributed as $bel(x_t)$, a new observation z_{t+1}, and a new action u_{t+1}, the particle filter generates a new weighted sample set that approximates $bel(x_{t+1})$.

The Particle Filtering (PF) algorithm proceeds in three steps:

(a) **Prediction step**: sample a new set of samples $x_{t+1}^{(i)}$, $i = 1, \ldots, N$, from a proposal distribution $g(x_{t+1}|x_t^{(i)}, z_{t+1}, u_{t+1})$.

(b) **Weighting step**: assign to each sample $x_{t+1}^{(i)}$ the weight:

$$w_{t+1}^{(i)} = w_t^{(i)} \frac{p(z_{t+1}|x_{t+1}^{(i)}) p(x_{t+1}^{(i)}|x_t^{(i)}, u_{t+1})}{g(x_{t+1}^{(i)}|x_t^{(i)}, z_{t+1}, u_{t+1})}$$

(c) **Resampling**: if the variance of the sample weights exceeds a certain threshold, resample with replacement, from the sample set, with probability proportional to $w_{t+1}^{(i)}$ and set the weights to 1.

A common simplification is the *bootstrap filter*, where the proposal distribution is the state transition model $g(x_{t+1}|x_t^{(i)}, z_{t+1}, u_{t+1}) = p(x_{t+1}|x_t^{(i)}, u_{t+1})$ and the weight simplifies to $w_{t+1}^{(i)} = w_t^{(i)} p(z_{t+1}|x_{t+1}^{(i)})$.

5 DCPF: a particle filter for dynamic distributional clauses

We now develop a particle filter for a set of dynamic distributional clauses that define the prior distribution, state transition model and observation model, (cf. Sect. 4.1). Throughout this development, we only consider the bootstrap filter for simplicity, but other proposal distributions are possible.

5.1 Filtering algorithm

The basic relational particle filter applies the same steps as the classical particle filter sketched in Sect. 4.2 and employs the forward reasoning procedure for distributional clauses sketched in Sect. 2. Each sample $x_t^{(i)}$ will be a complete possible world at time t. Working with complete worlds is computationally expensive and may lead to bad performance. Therefore, we shall work with samples that are partial worlds as in the static case. The resulting framework, that we shall now introduce, is called the Distributional Clauses Particle Filter (DCPF).

Starting from a DDC program $\mathbb{P}$, weighted partial samples $\{(x_t^{P(i)}, w_t^{(i)})\}$, a new observation list $z_{t+1} = \{z_{t+1}^j = v_j\}$, and a new action u_{t+1}, the DCPF filtering algorithm performs the weighting and prediction steps from time t to time $t+1$ expanding the partial samples as shown in Fig. 1. The new set of samples is $\{(\hat{x}_{t+1}^{P(i)}, w_{t+1}^{(i)})\}$. The DCPF filtering algorithm is the following:

- **Step (1)**: expand the partial sample to compute $w_{t+1}^{(i)} = w_t^{(i)} p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$
- Resampling (if necessary)
- **Step (2)**: complete the prediction step (a)

Step (1) performs the weighting step (b) and implicitly (part of) the prediction step (a): it computes the weight $w_{t+1}^{(i)} = w_t^{(i)} p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$ calling EvalSampleQuery$(z_{t+1}, x_t^{P(i)})$. EvalSampleQuery will automatically sample relevant variables at time $t+1$ and t until $p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$ is computable. If there are no observations, Step (1) is skipped, and the weights remain unchanged. At this point each sample has a new weight $w_{t+1}^{(i)}$, and resampling can be performed. For resampling we use a minimum variance sampling algorithm (Kitagawa 1996), but other methods are possible.

Step (2) performs the prediction step for variables that have not yet been sampled because they are not directly involved in the weighting step. The algorithm queries the head of any DC clause in the state transition model (intra-time clauses **excluded**), thus it evaluates the body recursively. Whenever the body is true for a substitution θ, the variable-distribution pair $r_{t+1}\theta \sim \mathcal{D}\theta$ is added to the sample. Avoiding sampling is beneficial for performance, as discussed in Sect. 3.1. Step (2) is necessary to make sure that the partial sample at the previous time step t can be safely forgotten, as we shall discuss in the next section. After Step (2) the set of partial samples $\hat{x}_{t+1}^{P(i)}$ with weights $w_{t+1}^{(i)}$ approximates the new belief $bel(x_{t+1})$.

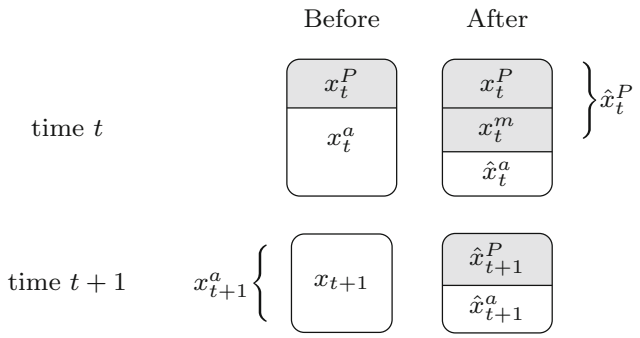

Fig. 1 Sample partition, before (*left*) and after (*right*) the filtering algorithm. Initially x_{t+1} is not sampled, therefore $x_{t+1}^a = x_{t+1}$ and $x_{t+1}^P = \emptyset$. The inference algorithm samples variables $x_t^m \subseteq x_t^a, x_{t+1}^m \subseteq x_{t+1}^a$ and adds them respectively to x_t^P and x_{t+1}^P. Indeed, $\hat{x}_t^P = x_t^P \cup x_t^m, \hat{x}_t^a = x_t^a \setminus x_t^m, \hat{x}_{t+1}^P = x_{t+1}^P \cup x_{t+1}^m = x_{t+1}^m$, $\hat{x}_{t+1}^a = x_{t+1}^a \setminus x_{t+1}^m$

In the classical particle filter resampling is the last step (c). In contrast, DCPF performs resampling before completing the prediction step (i.e., before Step (2)). This is loosely connected to auxiliary particle filters that perform resampling before the prediction step (Whiteley and Johansen 2010). Anticipating resampling is beneficial because it reduces the variance of the estimation.

To answer a query q_{t+1}, it suffices to call $(w_q^{(i)}, \hat{x}_{q_{t+1}}^{P(i)}) \leftarrow \textsc{EvalSampleQuery}(q, \hat{x}_{t+1}^{P(i)})$ for each sample and use formula (5) where $w_e^{(i)}$ is replaced by $w_{t+1}^{(i)}$. After querying, the partial samples $\hat{x}_{q_{t+1}}^{P(i)} \supseteq \hat{x}_{t+1}^{P(i)}$ are discarded, i.e., the partial samples remain $\hat{x}_{t+1}^{P(i)}$. This improves the performance, indeed querying does not expand $\hat{x}_{t+1}^{P(i)}$.

Example 9 Consider an extension of Example 8, where the next position of the object after a tap action is the current position plus a displacement and plus Gaussian noise. The latter two parameters depend on its type and the material of the object below it. We consider a single axis for simplicity.

$$\texttt{pos(ID)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_{\texttt{t}}), 0.01) \leftarrow \texttt{not}(\texttt{tap(ID)}_{\texttt{t+1}}). \tag{17}$$

$$\texttt{pos(ID)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_{\texttt{t}}) + 0.3, 0.04) \leftarrow \texttt{type(ID, cube)}, \texttt{on(ID, B)}_{\texttt{t}}, \texttt{material(B, wood)}, \texttt{tap(ID)}_{\texttt{t+1}}. \tag{18}$$

$$\texttt{pos(ID)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_{\texttt{t}}) + 0.2, 0.02) \leftarrow \texttt{type(ID, cube)}, \texttt{on(ID, B)}_{\texttt{t}}, \texttt{material(B, fabric)}, \texttt{tap(ID)}_{\texttt{t+1}}. \tag{19}$$

$$\texttt{pos(ID)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_{\texttt{t}}) + 1, 0.1) \leftarrow \texttt{type(ID, ball)}, \texttt{tap(ID)}_{\texttt{t+1}}. \tag{20}$$

$$\texttt{obsPos(ID)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_{\texttt{t+1}}), 0.01). \tag{21}$$

$$\texttt{type(1, cube). type(2, ball). type(3, table). material(3, wood).} \tag{22}$$

We define $\texttt{on(A, B)}_{\texttt{t}}$ from the z position of A and B. A is on B when A is above B and the distance is lower than a threshold. We omit the clause for brevity.

To understand the filtering algorithm let us consider Step (1) for the observation $\texttt{obsPos(1)}_{\texttt{t+1}} \sim= 2.5$ (there is no observation for object 2), and action $\texttt{tap(1)}_{\texttt{t+1}}$. Let us assume $x_t^{P(i)} = \{\texttt{pos(1)}_{\texttt{t}} = 2, \texttt{pos(2)}_{\texttt{t}} = 5\}$. The sample before and after the filter-

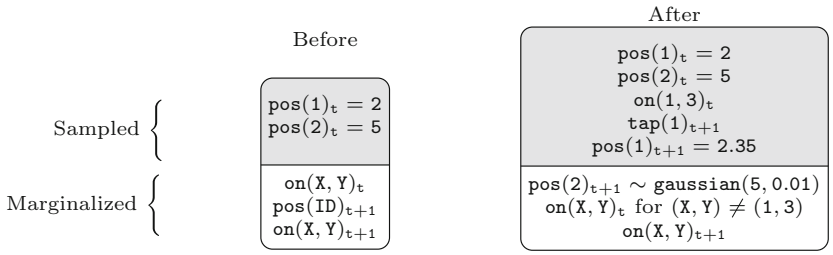

Fig. 2 A partial sample for Example 9, before (*left*) and after (*right*) the filtering algorithm

ing step is shown in Fig. 2. The algorithm tries to prove $\texttt{obsPos(1)}_{t+1} \sim\!= 2.5$. Rule 2b applies for DC clause (21) that defines $\texttt{obsPos(1)}_{t+1}$ for $\theta = \{\texttt{ID} = 1\}$. The algorithm tries to prove the body and the variables in the distribution recursively, that is $\texttt{pos(1)}_{t+1}$. The latter is not in the sample and rule 2b applies for DC clause (17) with $\theta = \{\texttt{ID} = 1\}$. The proof fails, therefore it backtracks and applies rule 2b to (18). Its body is true assuming that $\texttt{on(1, 3)}_t$ succeeds (and added to the sample). Thus, $\texttt{pos(1)}_{t+1}$ will be sampled from $\texttt{gaussian(2.3, 0.04)}$ and added to the sample (rule 3b). Deterministic facts in the background knowledge, such as $\texttt{type(1, cube)}$, are common to all samples; therefore, they are not added to the sample. At this point $p(\texttt{obsPos(1)}_{t+1} \sim\!= 2.5|\hat{x}_{t+1}^{P(i)})$ is given by (21). This is equivalent to applying rule 3a that imposes the query to be true and updates the weight.

Step (1) is complete, let us consider Step (2). The algorithm queries all the variables in the head of a clause in the state transition model, in this case $\texttt{pos(ID)}_{t+1}$. This is necessary to propagate the belief for variables not involved in the weighting process, such as $\texttt{pos(2)}_{t+1}$. The query $\texttt{pos(ID)}_{t+1} \sim\!= \texttt{Val}$ succeeds for $\texttt{ID} = 2$ applying (17), and $\texttt{pos(2)}_{t+1} \sim \texttt{gaussian(5, 0.01)}$ is added to the sample. The algorithm backtracks looking for alternative proofs of q, there are none, so the procedure ends. In the next step $\texttt{pos(2)}_{t+1}$ is required for $\texttt{pos(2)}_{t+2}$, so $\texttt{pos(2)}_{t+1}$ will be sampled from the distribution stored in the sample. Note that $\texttt{on(A, B)}_t$ is evaluated selectively. Any other relation or random variable eventually defined in the program remains marginalized. For example, any relation that involves object 2 is not required (e.g., $\texttt{on(2, B)}_t$).

5.2 Avoiding backinstantiation

We showed that lazy instantiation is beneficial to reduce the number of variables to sample and to improve the precision of the estimation in the static case. However, to evaluate a query at time t in dynamic models, the algorithm might need to instantiate variables at previous steps, sometimes even at time 0. We call this **backinstantiation**. This requires one to store the entire sampled trajectory $x_{1:t}^{P(i)}$, which may have a negative effect on performance. If we are interested in filtering, this is a waste of resources.

We shall now show that the described filtering algorithm performs lazy instantiation over time and avoids backinstantiation. We will first derive sufficient conditions for avoiding backinstantiation in DDC, and then prove that these conditions hold for the DCPF algorithm.

Rao-Blackwellization. Let us assume that the complete world at time t can be written as $x_t = x_t^P \cup x_t^a$ (Fig. 1). Let us consider the following factorization:

$$
\begin{aligned}
bel(x_t^a, x_{1:t}^P) &= p(x_t^a, x_{1:t}^P | z_{1:t}, u_{1:t}) \\
&= p(x_t^a | x_{1:t}^P, z_{1:t}, u_{1:t}) p(x_{1:t}^P | z_{1:t}, u_{1:t}) \\
&= p(x_t^a | x_{1:t}^P, z_{1:t}, u_{1:t}) bel(x_{1:t}^P) \\
&\approx \sum_{i=1}^{N} p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t}) w_t^{(i)} \delta_{x_{1:t}^{P(i)}}(x_{1:t}^P).
\end{aligned}
$$

In the particle filtering literature this is called Rao–Blackwellization (Doucet et al. 2000), where $bel(x_{1:t}^P)$ is approximated as a set of weighted samples[2] $\sum_i w^{(i)} \delta_{x_{1:t}^{P(i)}}(x_{1:t}^P)$ ($\delta_v(x)$ is the Dirac delta function centered in v), while the posterior distribution of x_t^a is available in closed form.

Rao–Blackwellized particle filters (RBPF) described in the literature, adopt a fixed and manually defined split of $x_t = x_t^P \cup x_t^a$. In contrast, our approach exploits the language and its inference algorithm to perform a dynamic split that may differ accross samples, as described for the static case in Sect. 3.

Backinstantiation in the DCPF. One contribution in the DCPF is that it integrates RBPF and logic programming to avoid backinstantiation over variables $r \in x_{1:t-1}$. For this reason we are interested in performing a filtering step determining the smallest partial samples that approximate the new belief $bel(x_{t+1})$ and are d-separated from the past. To avoid backinstantiation we require that $p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t})$ is a known distribution for each sample i or at most parametrized by $x_t^{P(i)}$: $p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t}) = f_t^{(i)}(x_t^a; x_t^{P(i)})$. Note that the latter equation does not make any independence assumptions: $f_t^{(i)}$ is a probability distribution that incorporates the dependency of previous states and observations and can be different in each sample.

Formally, starting from a partial sample $x_{1:t}^{P(i)}$ with weight $w_t^{(i)}$ sampled from $p(x_{1:t}^P | z_{1:t}, u_{1:t})$, a new observation z_{t+1}, a new action u_{t+1}, and the distribution $p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t})$, we look for the **smallest** partial sample $\hat{x}_{1:t+1}^{P(i)} = \{x_{1:t-1}^{P(i)}, \hat{x}_t^{P(i)}, \hat{x}_{t+1}^{P(i)}\}$ with $x_t^{P(i)} \subseteq \hat{x}_t^{P(i)}$, such that $\hat{x}_{1:t+1}^{P(i)}$ with weight $w_{t+1}^{(i)}$ is distributed as $p(\hat{x}_{1:t+1}^P | z_{1:t+1}, u_{1:t+1})$ and $p(x_{t+1}^a | \hat{x}_{1:t+1}^{P(i)}, z_{1:t+1}, u_{1:t+1})$ is a probability distribution available in closed form. Even though the formulation considers the entire sequence $x_{1:t+1}$, to estimate $bel(x_{t+1})$ the previous samples $\{x_{1:t-1}^{P(i)}, \hat{x}_t^{P(i)}\}$ can be forgotten.

D-separation conditions. We will now describe sufficient conditions that guarantee d-separation and thus avoid backinstantiation; then we will show that these conditions hold for the DCPF filtering algorithm. The belief update is performed by adopting RBPF steps.

Starting from $x_t^{P(i)}$, $w_t^{(i)}$, $p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t}) = f_t^{(i)}(x_t^a; x_t^{P(i)})$ and a new observation z_{t+1}, let us expand $x_t^{P(i)}$ sampling random variables $r_t \in var(x_t^a)$ from $f_t^{(i)}(x_t^a; x_t^{P(i)})$, and $r_{t+1} \in var(x_{t+1})$ from the state transition model $p(r_{t+1} | x_t^{P(i)})$ defined by DDC clauses, until the expanded sample $\{\hat{x}_t^{P(i)}, \hat{x}_{t+1}^{P(i)}\}$ and the remaining $\hat{x}_t^a, \hat{x}_{t+1}^a$ satisfy the following **conditions**:

1. the partial interpretation $\hat{x}_{t+1}^{P(i)}$ does not depend on the marginalized variables x_t^a: $p(\hat{x}_{t+1}^{P(i)} | \hat{x}_t^{P(i)}, \hat{x}_t^a, u_{t+1}) = p(\hat{x}_{t+1}^{P(i)} | \hat{x}_t^{P(i)}, u_{t+1})$;

[2] The Monte-Carlo approximation replaces a distribution with an empirical distribution given by a set of (weighted) samples. If the distribution is continuous the empirical distributon is described as a sum of Dirac delta centered in the samples.

2. the weighting function does not depend on the marginalized variables: $p(z_{t+1}|x_{t+1}^{(i)}) = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$
3. $p(\hat{x}_{t+1}^{a}|\hat{x}_{1:t+1}^{P(i)}, z_{1:t+1}, u_{1:t+1}) = f_{t+1}^{(i)}(\hat{x}_{t+1}^{a}; \hat{x}_{t+1}^{P(i)})$ is available in closed form.

Condition 1 is a common simplifying assumption in RBPF (Doucet et al. 2000), while condition 2 is not strictly required; however it simplifies the weighting and the computation of $f_{t+1}^{(i)}$. In some cases condition 2 can be removed, for example for discrete or linear gaussian models (using Kalman Filters). Condition 3 guarantees that the previous samples $\{x_{1:t-1}^{P(i)}, \hat{x}_{t}^{P(i)}\}$ can be forgotten to estimate $bel(x_\tau)$ for $\tau \geq t+1$.

Theorem 2 *Under the d-separation conditions 1,2,3 the samples* $\{\hat{x}_{t+1}^{P(i)}, f_{t+1}^{(i)}(\hat{x}_{t+1}^{a}; \hat{x}_{t+1}^{P(i)})\}$ *with weights* $\{w_{t+1}^{(i)}\}$ *approximate* $bel(x_{t+1})$*, with*

$$w_{t+1}^{(i)} \propto p(z_{t+1}|\hat{x}_{t+1}^{P(i)}) w_t \text{ and}$$

$$f_{t+1}^{(i)}(\hat{x}_{t+1}^{a}; \hat{x}_{t+1}^{P(i)}) = p(\hat{x}_{t+1}^{a}|\hat{x}_{1:t+1}^{P(i)}, z_{1:t+1}, u_{1:t+1}) = \int_{\hat{x}_t^a} p(\hat{x}_{t+1}^{a}|\hat{x}_{t}^{a}, \hat{x}_{t}^{P(i)}, \hat{x}_{t+1}^{P(i)}) f_t^{(i)}(\hat{x}_t^a; \hat{x}_t^{P(i)}) d\hat{x}_t^a. \quad (23)$$

If $\hat{x}_{t+1}^{a}$ *does not depend on* $\hat{x}_t^a$*, then* $f_{t+1}^{(i)}(\hat{x}_{t+1}^{a}; \hat{x}_{t+1}^{P(i)}) = p(\hat{x}_{t+1}^{a}|\hat{x}_{t}^{P(i)}, \hat{x}_{t+1}^{P(i)})$.

Proof The formulas are derived from RBPF (for the bootstrap filter) for which:

$$f_{t+1}^{(i)}(\hat{x}_{t+1}^{a}; \hat{x}_{t+1}^{P(i)}) = \frac{1}{\eta} p(z_{t+1}|x_{t+1}^{(i)}) \int_{\hat{x}_t^a} p(\hat{x}_{t+1}^{a}|\hat{x}_{t}^{a}, \hat{x}_{t}^{P(i)}, \hat{x}_{t+1}^{P(i)}) f_t^{(i)}(\hat{x}_t^a; \hat{x}_t^{P(i)}) d\hat{x}_t^a$$

$$w_{t+1}^{(i)} \propto \eta \cdot w_t^{(i)},$$

where $\eta = p(z_{t+1}|\hat{x}_{0:t+1}^{P(i)}, z_{1:t})$. Condition 2 makes $p(z_{t+1}|x_{t+1}^{(i)}) = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$ and $\eta = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$, simplifying the formulas. □

Step (1) in the filtering algorithm (Sect. 5.1) guarantees condition 1 and 2, while Step (2) guarantees condition 3. For a proof sketch see Theorems 5 and 6 in "Theorems" in the Appendix. Indeed, EvalSampleQuery used in Step (1) and (2) will never need to sample variables at time $t-1$ or before, because the belief distribution of marginalized variables $r_t \in \hat{x}_t^a$ is $f_t^{(i)}(x_t^a; \hat{x}_t^{P(i)})$ are available in closed form and (eventually) parameterized by $\hat{x}_t^{P(i)}$, while $r_{t+1} \in \hat{x}_{t+1}^a$ are sampled from the state transition model. After Step (2) any $r_{t+1} \in \hat{x}_{t+1}^a$ is derivable from $\hat{x}_{t+1}^{P(i)}$ together with the DDC program. These conditions avoid backinstantiation during filtering or query evaluation, thus previous partial states $x_{0:t}^{P(i)}$ can be forgotten.

Step (2) avoids computing the integral (23). The integral is approximated with a single sample, or equivalently the partial sample is expanded until $\hat{x}_{t+1}^a$ does not depend on marginalized $\hat{x}_t^a$, for which $f_{t+1}^{(i)}(\hat{x}_{t+1}^{a}; \hat{x}_{t+1}^{P(i)}) = p(\hat{x}_{t+1}^{a}|\hat{x}_{t}^{P(i)}, \hat{x}_{t+1}^{P(i)})$ is derivable from the DDC program. In detail, for each $r_{t+1}\theta \in \hat{x}_{t+1}^a$ Step (2) stores $r_{t+1}\theta \sim \mathcal{D}\theta$, where $\mathcal{D}\theta = p(r_{t+1}\theta|\hat{x}_t^{P(i)}, \hat{x}_{t+1}^{P(i)})$, e.g., $\texttt{pos(2)}_{\texttt{t+1}} \sim \texttt{Gaussian(5, 0.01)} = f_{t+1}^{(i)}(\texttt{pos(2)}_{\texttt{t+1}})$ as shown in Fig. 2. Such distribution is not parametrized and it is generally different in each sample. In contrast, $p(\texttt{size(A)}_{\texttt{t}}|x_{1:t}^{P(i)}, z_{1:t}, u_{1:t}) = Gaussian([\text{if } \texttt{type(A, ball)} \text{ then } \mu = 1; \text{ else } \mu = 2], 0.1) = f_t^{(i)}(\texttt{size(A)}_{\texttt{t}}; x_t^{P(i)})$ is a distribution parametrized by other variables in $x_t^{P(i)}$. It is sufficient to store this parametric function once, using DDC clauses,

instead of storing each distribution separately for each sample. Similarly, the DDC clause that defines $\texttt{on(A, B)}_t$ from object positions at time t is sufficient to represent all marginalized facts $\texttt{on(a, b)}_t$ not in $x_t^{P(i)}$. In general, intra-time DDC clauses can represent distributions of marginalized variables for an unspecified number of objects. Thus, Step (2) does not need to query variables defined by intra-time clauses, as proved in Theorem 6, "Theorems" in the Appendix.

Step (2) could be improved applying (23) whenever possible. Moreover, if there is a set of variables that has the same prior and transition model, the belief update (23) can be performed once for all whole set. Whenever a variable is required, it will be sampled. This does not require bounding the number of such sets of variables, and it can be considered as a simple form of lifted belief update. In some cases the belief $f_t^{(i)}(x_t^a; x_t^{P(i)})$ can be directly specified for any time t, we call this precomputed belief. As lifted belief update, this is applicable to an unbounded set of variables, and avoids unnecessary sampling.

Theorem 3 *DCPF has a space complexity per step and sample bounded by the size of the largest partial state $x_t^{P(i)}$, together with $f_t^{(i)}(x_t^a; x_t^{P(i)})$.*

Proof (sketch) The filtering algorithm proposed for DCPF avoids backinstantiation, therefore the space complexity is bounded by the dimension of the state space at time t. A tighter bound is the size of the largest partial state. □

5.3 Limitations

We will now describe the limitations of the proposed algorithms.

Lazy instantiation is beneficial only when there are facts or random variables that are irrelevant during inference. For example, this is true when the model includes background knowledge that is not entirely required for a query. In fully-connected models or when the entire world is relevant for a query, lazy instantiation is useless. Nonetheless, the proposed method generalizes LW, thus it can be beneficial even in the described worse cases.

The above issues apply also to inference in dynamic models, but the latter raises additional issues for filtering in particular. One is the curse of dimensionality that produces poor results for high-dimensional state spaces, or equivalently it requires a huge number of samples to give acceptable results. There are some solutions in the literature (e.g., factorising the state space Ng et al. 2002). In DCPF this problem can be alleviated using a suboptimal proposal distribution (see "Proposal distribution" in the Appendix for details).

6 Online parameter learning

So far we assumed that the model used to perform state estimation is known. In practice, it may be hard to determine or to tune the parameters manually, and therefore the question arises as to whether we can learn them. We will first review online parameter learning in classical particle filters, then we will show how to adapt those methods in DCPF.

6.1 Learning in particle filters

A simple solution to perform state estimation and parameter learning with particle filters consists of adding the static parameters to the state space: $\bar{x}_t = \{x_t, \theta\}$. The posterior distribution $p(\bar{x}_t|z_{1:t})$ is then described as a set of samples $\{x_t^{(i)}, \theta^{(i)}\}$. However, this solution produces poor results due to the following *degeneracy problem*. As the parameters are sampled in the first step and left unchanged (since they are static variables), after a few steps

the parameter samples $\theta^{(i)}$ will degenerate to a single value due to resampling. This value will remain unchanged regardless of incoming new evidence. Limiting or removing resampling is not a good solution, because it will produce poor state estimation results. Better strategies have been proposed and are summarized in Kantas et al. (2009). We focus on two simple techniques with limited computational cost: artificial dynamics (Higuchi 2001) and resample-move (Gilks and Berzuini 2001). Both methods introduce diversity among the samples to solve the described degeneration problem.

The first method adds **artificial dynamics** to the parameter θ: $\theta_{t+1} = \theta_t + \epsilon_{t+1}$, where ϵ_{t+1} is artificial noise with a small and decreasing variance over time. This strategy has the advantage to be simple and fast, nonetheless it modifies the original problem and requires tuning (Kantas et al. 2009). We will show that this technique is suitable for the scenarios considered in this paper (for a limited number of parameters).

The second method is **resample-move** that adds a single MCMC step to rejuvenate the parameters in the samples. There are several variants of this technique, the most notable are Storvik's filter (Storvik 2002) and Particle Learning by Carvalho et al. (2010). To understand these approaches, consider the following factorization of the joint distribution of interest:

$$\begin{aligned} p(x_{0:t}, \theta | z_{1:t}) &= \frac{p(x_{0:t}, \theta, z_t | z_{1:t-1})}{p(z_t | z_{1:t-1})} \\ &= p(\theta | x_{0:t-1}, z_{1:t-1}) p(x_{0:t-1} | z_{1:t-1}) \frac{p(z_t | x_t, \theta) p(x_t | x_{t-1}, \theta)}{p(z_t | z_{1:t-1})}. \end{aligned}$$

In addition to the standard propagation and weighting steps, both algorithms perform a Gibbs sampling step that samples a new parameter value $\theta_t^{(i)}$ from the distribution $p(\theta | x_{0:t}^{(i)}, z_{1:t}) = p(\theta | s_t^{(i)})$ where $s_t^{(i)}$ captures the sufficient statistics of the distribution. $p(\theta | x_{0:t}, z_{1:t})$ is recursively updated as follows:

$$\begin{aligned} p(\theta | s_t) &= p(\theta | x_{0:t}, z_{1:t}) \\ &\propto p(z_t, x_t, \theta | x_{0:t-1}, z_{1:t-1}) \\ &= p(\theta | x_{0:t-1}, z_{1:t-1}) p(z_t | x_t, \theta) p(x_t | x_{t-1}, \theta) \\ &= p(\theta | s_{t-1}) p(z_t | x_t, \theta) p(x_t | x_{t-1}, \theta) \end{aligned} \tag{24}$$

This leads to a deterministic sufficient statistics update $s_t = S(s_{t-1}, x_t, x_{t-1}, z_t)$.
Storvik's filter algorithm goes as follows:

- propagate: $x_t^{(i)} \sim g(x_t | x_{t-1}^{(i)}, \theta_{t-1}^{(i)}, z_t)$,
- resample samples with weights: $w_t^{(i)} = \frac{p(z_t | x_t^{(i)}, \theta_{t-1}^{(i)}) p(x_t^{(i)} | x_{t-1}^{(i)}, \theta_{t-1}^{(i)})}{g(x_t^{(i)} | x_{t-1}^{(i)}, \theta_{t-1}^{(i)}, z_t)}$,
- propagate sufficient statistics: $s_t^{(i)} = S(s_{t-1}^{(i)}, x_t^{(i)}, x_{t-1}^{(i)}, z_t^{(i)})$, and
- sample $\theta_t^{(i)} \sim p(\theta | s_t^{(i)})$.

The Particle Learning proposed by Carvalho et al. (2010) is an optimization of Storvik's filter since it adopts the auxiliary particle filter (Pitt and Shephard 1999) and an optimal proposal distribution g. Resample-move strategies do not change the problem as in the artificial dynamics, and have been proven to be successful for several classes of problems (Carvalho et al. 2010; Carvalho et al. 2010; Lopes et al. 2010). However, they suffer from the sufficient statistics degeneracy problem that can produce an increasing error in the parameter posterior distribution (Andrieu et al. 2005).

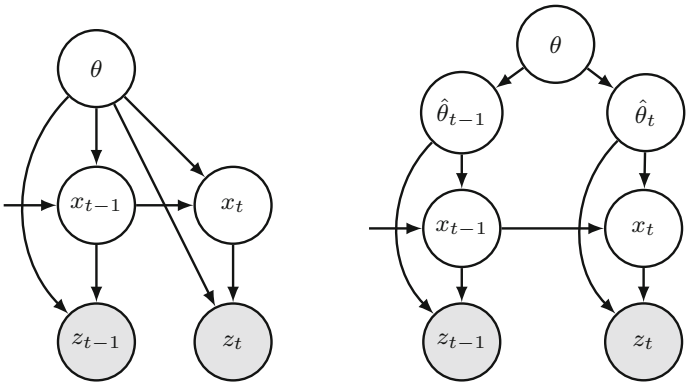

Fig. 3 *Left* HMM-like dynamic model parameterized by θ. *Right* modified version used to apply a Storvik's filter variant in DCPF

6.2 Online parameter learning for DCPF

We now propose an integration of the mentioned learning methods in DCPF. The main contribution is to adapt artificial dynamics and the Storvik's filter for DCPF and allow learning of a number of parameters defined at run-time. Indeed, the relational representation allows to define an unbounded set of parameters to learn, e.g., the size of each object `size(ID)`. The number of objects and thus parameters to learn is not necessarily known in advance.

Artificial dynamics in DCPF. To describe the integration of artificial dynamics in DCPF we consider an object tracking scenario called Learnsize (Sect. 7.4), in which the parameters to learn are the sizes of all objects. We defined a uniform prior: $\texttt{size(ID)}_0 \sim \texttt{uniform(0, 20)}$. Since the number of objects is not defined in advance we can directly define the size distribution at time t for any $\texttt{size(ID)}_\texttt{t}$ not yet sampled: $\texttt{size(ID)}_\texttt{t} \sim \texttt{uniform(0, 20)}$ (i.e., $f_t^{(i)}(x_t^a; x_t^{P(i)})$ is directly defined for not sampled $\texttt{size(ID)}_\texttt{t}$). Whenever the size of an object `x` (not yet sampled) is needed for inference, $\texttt{size(x)}_\texttt{t}$ is sampled for the above rule with no need to perform backinstantiation. While the transition model defines the artificial dynamics: $\texttt{size(ID)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{size(ID)}_\texttt{t}), \bar{\sigma}^2/\texttt{T}^\texttt{X})$, where `T` is the time step, `X` is a fixed exponent (set to 1 in this paper) and $\bar{\sigma}^2$ is a constant that represents the initial variance. Initially the variance is high, thus the particle filter can "explore" the parameter space, after some steps the variance decreases in the hope that the parameter converges to the real value.

Storvik's filter in DCPF. Equation (24) shows how to update the parameter posterior and then the sufficient statistics for each sample. However, this formulation needs a conjugate prior for the parameter likelihood. For a complex distribution this may be hard. We developed a variant of Storvik's filter to overcome this problem. In detail, we add $\hat{\theta}_t$ to the state that represents the currently sampled "parameter" value, while θ is the parameter to estimate, e.g., the mean of $\hat{\theta}_t$ (Fig. 3 on the right). We also assume the state x_t and the observations depend only on $\hat{\theta}_t$: $p(x_t|x_{t-1}, \hat{\theta}_t, \theta) = p(x_t|x_{t-1}, \hat{\theta}_t)$ and $p(z_t|x_t, \hat{\theta}_t, \theta) = p(z_t|x_t, \hat{\theta}_t)$. Thus, the posterior becomes:

$$p(x_{0:t}, \theta, \hat{\theta}_{0:t}|z_{1:t}) \propto p(z_t, x_{0:t}, \theta, \hat{\theta}_{0:t}|z_{1:t-1}) =$$
$$= p(z_t, x_t, \hat{\theta}_t, \theta, \hat{\theta}_{0:t-1}, x_{0:t-1}|z_{1:t-1}) =$$
$$= p(z_t|x_t, \hat{\theta}_t)p(x_t|x_{t-1}, \hat{\theta}_t)p(\hat{\theta}_t|\theta)p(\theta|\hat{\theta}_{0:t-1})p(\hat{\theta}_{0:t-1}, x_{0:t-1}|z_{1:t-1}).$$

Knowing that $p(\theta|\hat{\theta}_{0:t}, x_{0:t}, z_{1:t}) \propto p(x_{0:t}, \theta, \hat{\theta}_{0:t}|z_{1:t})$ we replace (24) with:

$$p(\theta|\hat{\theta}_{0:t}, x_{0:t}, z_{1:t}) \propto p(\hat{\theta}_t|\theta)p(\theta|\hat{\theta}_{0:t-1}) = p(\hat{\theta}_t|\theta)p(\theta|s_{t-1}).$$

Thus $p(\theta|\hat{\theta}_{0:t}, x_{0:t}, z_{1:t}) = p(\theta|\hat{\theta}_{0:t}) = p(\theta|s_t)$. At this point we can avoid sampling θ as required by the Storvik's filter, but sample $\hat{\theta}_t$ from the marginal distribution: $p(\hat{\theta}_t|s_{t-1}) = \int_\theta p(\hat{\theta}_t|\theta)p(\theta|s_{t-1})d\theta$.

For example, in the Learnsize scenario, for each object `ID` we have $\hat{\theta}_t = \texttt{cursize}_t(\texttt{ID})$ and the parameter to learn is $\theta = \texttt{size(ID)}$. For each object $p(\hat{\theta}_t|\theta)$ is defined as $\texttt{cursize}_t(\texttt{ID}) \sim \texttt{Gaussian}(\simeq\texttt{size(ID)}, \bar{\sigma}^2)$, where $\bar{\sigma}^2$ is a fixed variance. The conjugate prior of `size(ID)` is a Gaussian with hyperparameters $\mu_0(\texttt{ID}), \sigma_0^2(\texttt{ID})$: $\texttt{size(ID)} \sim \texttt{Gaussian}(\mu_0(\texttt{ID}), \sigma_0^2(\texttt{ID}))$. As explained $\theta = \texttt{size(ID)}$ need not be sampled, indeed $\hat{\theta}_t = \texttt{cursize}_t(\texttt{ID})$ is directly sampled from $p(\hat{\theta}_t|s_{t-1})$, i.e. $\texttt{cursize}_t(\texttt{ID}) \sim \texttt{Gaussian}(\mu_{t-1}(\texttt{ID}), \sigma_{t-1}^2(\texttt{ID}) + \sigma^2)$. For each `ID` the posterior $p(\theta|s_t)$ is a Gaussian as the prior, and the sufficient statistics $s_t = \mu_t(\texttt{ID}), \sigma_t^2(\texttt{ID})$ are computed using Bayesian inference. The posterior distribution of the parameters can become peaked in few steps, causing again a degeneration problem. This issue is mitigated reducing the influence of the evidence during the Bayesian update.

7 Experiments

This section answers the following questions:

(Q1) Does the EvalSampleQuery algorithm obtain the correct results?
(Q2) How do the DCPF and the classical particle filter compare?
(Q3) How do the DC and DCPF perform with respect to a representative state-of-the-art probabilistic programming language for static and dynamic domains?
(Q4) Is the DCPF suitable for real-world applications?
(Q5) How do the learning algorithms perform?

Among the several state-of-the-art probabilistic languages, BLOG (Milch et al. 2005) is a system that shares some similarities with DC and DCPF. For this reason, we compared the performance with BLOG for static domains and its dynamic extension DBLOG (de Salvo Braz et al. 2008) for temporal domains.

All algorithms were implemented in YAP Prolog and run on an Intel Core i7 3.3GHz for simulations and on a laptop Core i7 for real-world experiments. To measure the error between the predicted and the exact probability for a given query, we compute the empirical standard deviation (STD). The average used to compute STD is the exact probability when available or the empirical average otherwise. We report STD 99% confidence intervals. Notice that those intervals refer to the uncertainty of the STD estimation, not to the uncertainty of the probability. If the number of samples is not sufficient to give an answer (e.g., all samples are rejected), a value is randomly chosen from 0 to 1. The results are averaged over 500 runs. In all the experiments we measure the CPU time ("user time" in the Unix "time" command). This makes a fair comparison between DC and DCPF (not parallelized in the

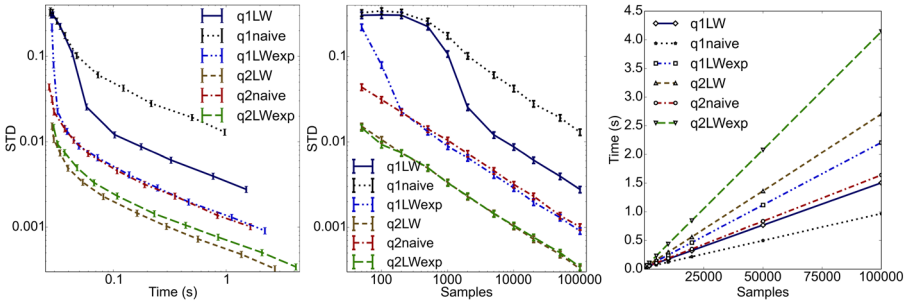

Fig. 4 Results for static inference with LW and without LW (naive). For LW we show results with (LWexp) and without query expansion. The queries are $\texttt{q1} = (\texttt{drawn(3)} \sim\!= \texttt{10})$ with evidence $((\texttt{drawn(1)} \sim\!= \texttt{9}, \texttt{drawn(2)} \sim\!= \texttt{9})\ \texttt{OR}\ (\texttt{drawn(1)} \sim\!= \texttt{10}, \texttt{drawn(2)} \sim\!= \texttt{10}))$ and $\texttt{q2} = (\texttt{drawn(1)} \sim\!= \texttt{X}, \texttt{drawn(2)} \sim\!= \texttt{X}, \texttt{color(X)} \sim\!= \texttt{black})$. The axes in (a) and (b) are in logarithmic scale. q1LWexp and q2naive partially overlap in (a) and (b); q2LWexp and q2LW overlap in (b)

current implementation) and (D)BLOG that often uses more than one CPU at the time. Time includes initialization: around $0.3s$ for (D)BLOG, $0.03s$ for DC and DCPF.

We first describe experiments in static domains, then in dynamic domains (synthetic and real-world scenarios).

7.1 Static domains

In the **first experiment** we tested the correctness of EVALSAMPLEQUERY for static inference with and without LW and query expansion (Q1) using Example 4 in Sect. 3.3. Figure 4 shows some results. The error (STD) converges to zero for all algorithms. EVALSAMPLEQUERY with LW (without query expansion) has a lower STD (Fig. 4b), but it is slower for the same number of samples (Fig. 4c). Nonetheless, the overhead is beneficial because for the same execution time the STD of LW is lower (Fig. 4a). Adding query expansion (Sect. 3.3.1) to LW has a computational cost. This is beneficial for query q1 (as defined in the caption of Fig. 4), allowing to exploit LW in disjunctions. Nonetheless, for query q2 the query expansion overhead is not compensated by an error reduction.

In the **second experiment** (Fig. 5) we considered an identity uncertainty domain[3] used in BLOG (Milch et al. 2005). The query considered is the probability that the second and third drawn balls are the same, given that the color of the drawn balls are respectively black, white, and white. We compared DC with BLOG. We consider several settings for DC: naive (EVALSAMPLEQUERY without LW), LW (EVALSAMPLEQUERY with LW, using formula (5) to compute $p(q|e)$), and LW2 where EVALSAMPLEQUERY estimates $p(q, e)$ and $p(e)$ independently using half of the available samples each. LW and LW2 are tested with and without query expansion (respectively LWexp and LW2exp). The results show that the error (STD), for the same number of samples or time, is lower for DC with LW (Fig. 5a, b). In particular, the lowest error is obtained with LW2exp. Any DC setting is faster than BLOG (Fig. 5c). The latter has an unexpected logarithmic-like behavior for a small number of samples. In addition, it seems that BLOG does not use LW in this domain, indeed the error is comparable with naive Monte Carlo for the same number of samples (Fig. 5b). In

[3] Available at https://github.com/BayesianLogic/blog/blob/master/example/balls/id-uncert-det.blog.

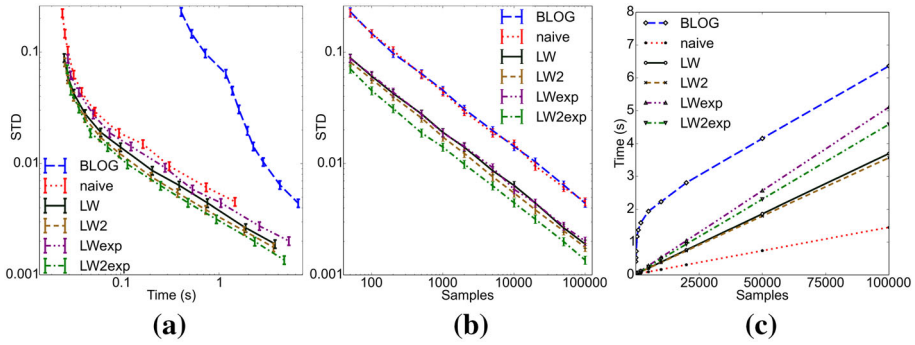

Fig. 5 Identity uncertainty domain used in Milch et al. (2005). The axes in (**a**) and (**b**) are in logarithmic scale. LW and LWexp overlap in (**b**); BLOG and naive overlap in (**b**); LW and LW2 overlap in (**c**)

contrast, as described in Sect. 3.3.2 EvalSampleQuery exploits LW in complex queries as equalities between random variables.

In the **third experiment** we considered continuous variables using Example 4 (Fig. 6). We queried the probability that the first drawn ball is made of wood, given that its size is 0.4. BLOG and naive MC failed to give an answer, while DC (LW) provides a probability. To compare with BLOG we had to consider an interval instead of a value ([0.39, 0.41]). Both DC and BLOG converge to 0.16, this confirms that DC works properly with continuous variables. Figure 6 shows the STD and time performance. EvalSampleQuery exploits LW also in this case, while BLOG needs evidence discretization to give an answer and does not exploit LW in this case. For this reason BLOG performs poorly. In this case query expansion does not provide an improvement. Another tested query is the probability that two drawn balls are the same, given that they have the same size. This probability is one, because the size has a continuous distribution, thus the probability of having two different balls with the same size is infinitely smaller than the probability of sampling the same ball; for this reason the balls must be the same. Again, DC provides the correct result, while BLOG does not provide an answer.

In the **fourth experiment** we considered the Indian GPA problem (Example 6). Most probabilistic programming languages are not able to handle this domain. In contrast, DC is able to give the correct results. For instance, it provides $p(\texttt{nation} \sim= \texttt{america} \mid \texttt{studentGPA} \sim= 4) = 1 \pm 0$ and $p(\texttt{nation} \sim= \texttt{america} \mid \texttt{studentGPA} \sim= 3.9) = 0.193 \pm 0.0035$ as expected, respectively in 0.33 and 0.37s with 10,000 samples. In this domain the query expansion is necessary.

From the above experiments we make the following comparison with BLOG. Given a query BLOG stacks variables that need to be sampled to answer the query, and uses LW to generate samples consistent with the evidence. This follows the lazy instantiation principle applied in EvalSampleQuery, nonetheless there are the following differences. BLOG exploits LW only for simple evidence statements of the form $\texttt{r} = \texttt{value}$, thus it performs worse than DC with complex queries described in Sect. 3.3.2, which the experiments confirm. Furthermore, BLOG is not always able to give an answer for complex queries or evidence containing continuous variables as shown above. In contrast, DC gives meaningful answers and exploits LW in a much wide range of queries. Finally, BLOG seems to be less suited than DC for real-time inference, because it is generally slower with higher variance.

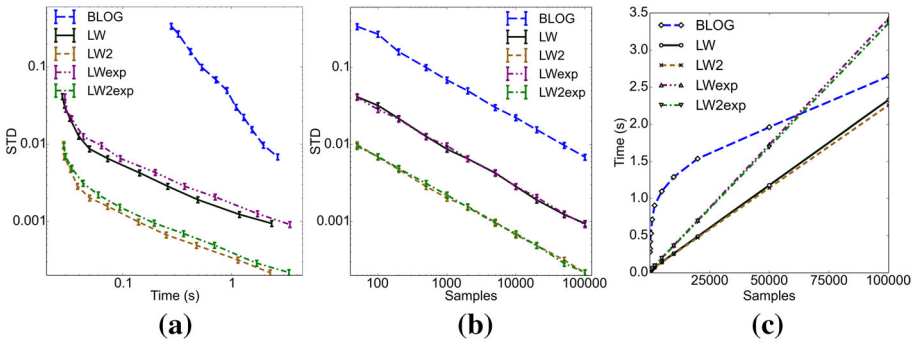

Fig. 6 Experiments with continuous evidence. The query is the probability that the first drawn ball is made of wood, given that its size is 0.4. BLOG requires evidence discretization. LW and LWexp overlap in (**b**); LW2 and LW2exp overlap in (**b**); LWexp and LW2exp overlap in (**c**)

7.2 Synthetic dynamic domains

We now answer questions Q2 and Q3 comparing the classical particle filter, DCPF and DBLOG in dynamic domains. In all dynamic experiments we disabled the query expansion because it is not necessary.

In this section we used a probabilistic Wumpus world [inspired by Russell and Norvig (2009)]. This is a discrete world with a two-dimensional grid of cells, that can be either free, a wall, or a pit. In one of the cells the horrible wumpus lives and each cell can contain gold. Each pit produces a breeze in the neighboring cells, and the wumpus produces a stench in the neighboring cells. The agent has to estimate the hidden state consisting of the wumpus' location, the state of each cell (free, wall or pit), as well as its own position in the maze. The agent has four stochastic 'move' actions: up, down, left, right, which change the position by 1 cell or lead to no change with a particular probability. Furthermore, the agent has noisy sensors to observe whether there is a breeze, a stench, or gold in the cell, and whether there are walls in the neighboring cells. We assume that the agent starts from position (0,0), therefore the cell (0,0) is free.

In the Wumpus domain we use a lifted belief update or precomputed belief. For example, if the belief at time t for each cell not in the partial sample is $\texttt{maze(X, Y)}_t \sim \texttt{finite([0.6 : free, 0.4 : wall])}$ and the state transition is $\texttt{maze(X, Y)}_{t+1} \sim \texttt{val}(\simeq\texttt{(maze(X, Y)}_t\texttt{)})$ (the next cell state is equal to the current cell state), the belief update can be done once for all the cells that have not been sampled yet. In this case the belief remains the same over time, thus, we can directly define the belief at time t without doing lifted belief update. If a cell $\texttt{maze(x, y)}_t$ is required it will be sampled, and the belief update for this cell will be performed by sampling. For non-sampled cells $\texttt{maze(X, Y)}_t \sim \texttt{finite([0.6 : free, 0.4 : wall])}$ is used instead. Lifted belief update and precomputed belief do not require that the size of the grid is specified.

Classical Particle Filter (Q2). The classical particle filter samples the entire state every step with a forward reasoning procedure.

In the **first experiment** (Wumpus1) there are no pits and no wumpuses. The goal is to compute the joint distribution of 3 cells state $(\texttt{maze(0, 2)}_t, \texttt{maze(1, 1)}_t, \texttt{maze(2, 0)}_t)$ given noisy gold and cell observations. The experiment consists of 3 steps (to keep exact inference feasible) with one or two observations per step. In the classical particle filter, we

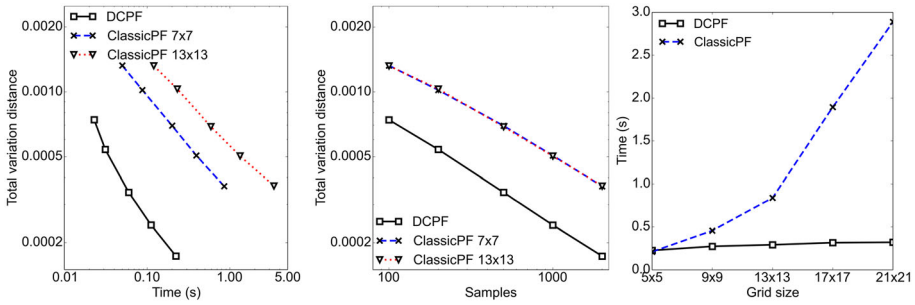

Fig. 7 Experiments. Total variation distance as a function of run-time (**a**) or the number of samples (**b**) for the probabilistic wumpus example. **c** Run-time for various grid sizes with 1000 samples. **a**, **b** are in logarithmic scale

need to limit the size of the grid in advance. In contrast, the DCPF estimates the borders of the maze itself. To measure the error between the predicted and the exact posteriors we use the total variation divergence (i.e., the sum of absolute differences averaged over runs). Figure 7a, b show that both algorithms provide correct results (Q1), but our DCPF produces lower errors and is faster when compared to the classical particle filter for the same number of samples (Fig. 7b). This is because DCPF reduces the number of tracked variables and therefore reduces both the variance in the sampling process and the execution time. As expected, the maze size of the classical particle filter affects the performance. The DCPF by contrast does not require a fixed grid size using a precomputed belief or lifted belief update. This makes DCPF more flexible and faster in comparison (Q2).

In the **second experiment** (Wumpus2) we used a wumpus world with one wumpus that produces stench sensed with a noisy sensor. We also model the agent's energy as a continuous variable that decreases over time with Gaussian noise. The probability to move is a function of the energy. We randomly generated worlds of different sizes (2 worlds per size, 5 runs per world), together with a sequence of 100 actions and observations (neighboring cells state, stench and energy). The sequence of actions and observations was used as the input to the particle filter (classic or DCPF) with 1000 samples. The model is too complex to compute exactly, therefore we focused on runtime evaluation. The results (Fig. 7c) show that DCPF is faster than the classical particle filter (Q2), because it reduces the sample size. For completeness, in the last step the average number of sampled variables in DCPF was 30.8, 40.6, 51.8, 53.1, 61.4 respectively for worlds 5x5, 9x9, 13x13, 17x17, 21x21.

DBLOG (Q3). Because (D)BLOG cannot fully cope with continuous evidence as DCPF, we now focus on a discrete case for a further comparison.

In the **third experiment** (Wumpus3) we used a Wumpus domain similar to Wumpus1 to compare DCPF with DBLOG. We executed 10 steps and queried $q = (\texttt{maze}(2,1)_t \sim= \texttt{free}, \texttt{maze}(0,1)_t \sim= \texttt{free}, \texttt{maze}(1,0)_t \sim= \texttt{free}, \texttt{gold}(1,1)_t \sim= \texttt{true})$; the error and time performance are shown in Fig. 8. The results highlight that DCPF has a slightly lower error than DBLOG for the same number of samples (Fig. 8b). In addition DCPF is faster for a small number of samples (Fig. 8c). For a large number of samples DBLOG becomes faster, probably because of the unoptimized prolog implementation of DCPF.

The **fourth experiment** (Wumpus4) is similar to the previous one (Wumpus3) where the evidence contains statements such as the observed cell on the left is equal to the observed

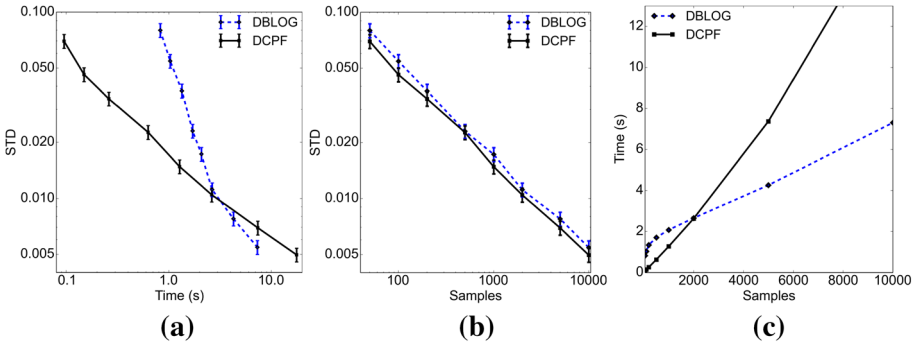

Fig. 8 Experiments (Wumpus3). Time refers to 10 steps. The axes in (**a**) and (**b**) are in logarithmic scale. For STD there is a 99% confidence interval

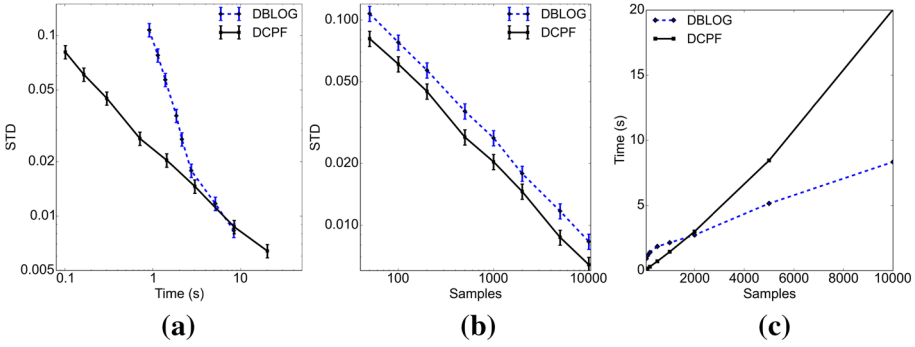

Fig. 9 Wumpus experiments with complex evidence (Wumpus4). The axes in (**a**) and (**b**) are in logarithmic scale. Time refers to 10 steps. For STD there is an 99% confidence interval

cell of the right. The results are shown in Fig. 9 and highlight that DCPF has a lower error for the same number of particles (Fig. 9b) because DCPF exploits LW with such complex evidence.

DBLOG does not currently implement Step (2) of our filtering algorithm[4]. This requires a workaround that consists of in manually querying all the variables that might be relevant for the given query, thus to fix the number of random variables (e.g., the size of the maze). In Wumpus3 and Wumpus4 this is avoided because the cell variables are static and they do not require belief updates. If the cell state changes over time, DBLOG needs to query all the the cells at each step. This makes DBLOG equivalent to a classical particle filter, where the size of the maze is fixed and sampled entirely. Thus, DBLOG becomes slow with high variance, while for DCPF we exploit lifted belief update. This is shown in the **fifth experiment** (Wumpus5, Fig. 10) where the cells state changes over time. It is problematic for DBLOG when the worlds may contain different numbers of random variables. It is not trivial to determine in advance which variables are relevant for a given query at time t. Indeed, every sample will have different variables, thus propagation has to be performed in a different way for each sample. In contrast, Step (2) in DCPF samples variables that are sufficient to guarantee d-separation, and those variables can be different in different samples because

[4] According to the following bug report https://github.com/BayesianLogic/blog/issues/330.

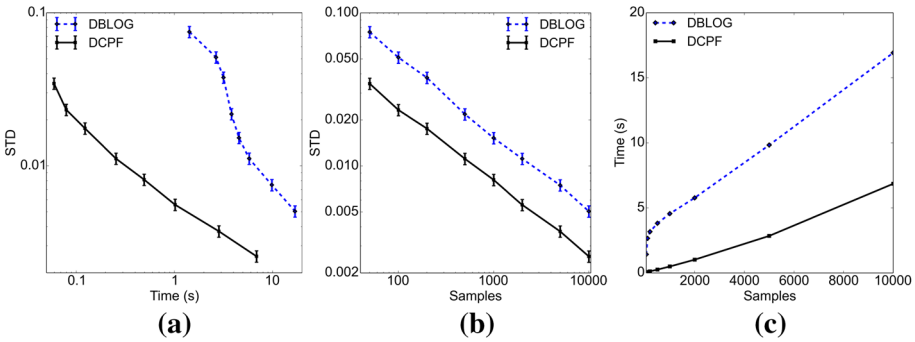

Fig. 10 Wumpus experiments with changing cells (Wumpus5). Time refers to 4 steps. For DBLOG the rows and columns cells from $[-3, 3]$ has been queried at each step. The axes in (**a**) and (**b**) are in logarithmic scale

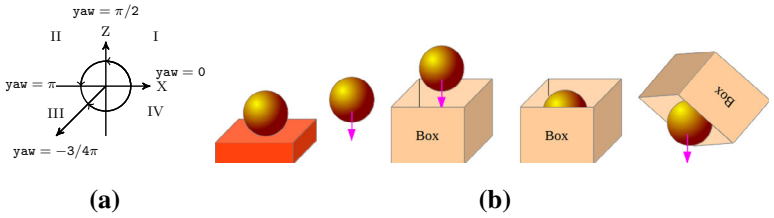

Fig. 11 **a** Yaw of an object. Yaw is positive in quadrants I and II. **b** Physical principles considered

of context-specific independences. This avoids backinstantiation with the possibility to use lifted belief updates and precomputed beliefs.

7.3 Real-world dynamic domains

The experiments shown so far are generated from synthetic data. We also ran experiments with real-world data (Q4). We considered two tracking scenarios called Packaging and Learnsize; the latter is used to evaluate parameter learning. The objects have markers for an easy detection with a camera (Fig. 14).

7.3.1 Packaging scenario

The goal of this scenario is to track objects moved by a human during a packaging activity with boxes (Fig. 12). The framework should be able to keep track of objects inside boxes. To solve this problem we defined a model in Dynamic Distributional Clauses where the **state** consists of the position, the velocity and orientation of all objects, plus the relations between them. The relations considered are left, right, near, on, and inside plus object properties such as color, type and size; whenever a new object is observed its pose is added to the state together with all the derived relations.
We modelled the following physical principles (Fig. 11b) in the **transition model**:

1. if an object is on top of another object, it cannot fall down;
2. if there are no objects under an object, the object will fall down until it collides with another object or the table;

3. an object may fall inside the box only if it is on the box in the previous step, is smaller than the box and the box is open-side up;
4. if an object is inside a box it remains in the box and its position follows that of the box as long as it is open-side up;
5. if the box is rotated upside down the objects inside will fall down with a certain probability.

As example consider the property 3: if an object A is not inside a box, is on top of a box B with rotation $\mathtt{yaw}_t(\mathtt{B}) > 0$ (i.e. open-side up, Fig. 11a) and the object A is smaller than the box B, then it falls inside B with probability 0.8 in the next step. This can be modelled by the following clause:

$$\begin{aligned}\mathtt{inside}_{t+1}(\mathtt{A},\mathtt{B}) \sim \mathtt{finite}([0.8:\mathtt{true}, 0.2:\mathtt{false}]) \leftarrow \\ \mathtt{not}(\mathtt{inside}_t(\mathtt{A},\mathtt{C}) \sim= \mathtt{true}), \mathtt{on}_t(\mathtt{A},\mathtt{B}), \mathtt{type}(\mathtt{B},\mathtt{box}), \\ \simeq(\mathtt{yaw}_t(\mathtt{B})) > 0, \mathtt{smaller}(\mathtt{A},\mathtt{B}).\end{aligned} \tag{25}$$

To model the position and the velocity of objects in free fall we use the rule:

$$\begin{aligned}\mathtt{pos\_vel}_{t+1}(\mathtt{A})_z \sim \mathtt{gaussian}\left(\begin{bmatrix}\simeq(\mathtt{pos}_t(\mathtt{A}))_z + \Delta t \cdot \simeq(\mathtt{vel}_t(\mathtt{A}))_z - 0.5g\Delta t^2 \\ \simeq(\mathtt{vel}_t(\mathtt{A}))_z - g\Delta t\end{bmatrix}, \Sigma\right) \\ \leftarrow \mathtt{not}(\mathtt{inside}_t(\mathtt{A},\mathtt{C}) \sim= \mathtt{true}), \mathtt{not}(\mathtt{on}_t(\mathtt{A},\mathtt{D})), \mathtt{held}_t(\mathtt{A}) \sim= \mathtt{false}.\end{aligned} \tag{26}$$

It states that if the object A is neither 'on' nor 'inside' any object, and is not held, the object will fall with gravitational acceleration g. For simplicity we specify only the position and velocity for the coordinate z. The variable $\mathtt{held}_t(\mathtt{A})$ indicates whether the object is held or not, let us assume the following distribution:

$$\mathtt{held}_t(\mathtt{A}) \sim \mathtt{finite}([0.6:\mathtt{true}, 0.4:\mathtt{false}]). \tag{27}$$

The measurement model is the product of Gaussian distributions around each object's position (thereby assuming i.i.d. measurements):

$$\mathtt{obsPos}_{t+1}(\mathtt{A}) \sim \mathtt{gaussian}(\simeq(\mathtt{pos}(\mathtt{A})_t), \Sigma_{obs}). \tag{28}$$

where $\mathtt{pos}(\mathtt{A})_t$ is the subvector of $\mathtt{pos\_vel}_t$ related to the x, y, z position, and Σ_{obs} is a fixed covariance matrix.

Furthermore, if A is inside B at time t, the relation holds at $t+1$ with probability close to one (clause omitted). The inside concept is transitive, therefore we defined a transitive inside relation $\mathtt{tr\_inside}_t(\mathtt{A},\mathtt{B})$ from $\mathtt{inside}_t(\mathtt{A},\mathtt{B})$:

$$\begin{aligned}\mathtt{tr\_inside}_t(\mathtt{A},\mathtt{B}) &\leftarrow \mathtt{inside}_t(\mathtt{A},\mathtt{B}) \sim= \mathtt{true}. \\ \mathtt{tr\_inside}_t(\mathtt{A},\mathtt{B}) &\leftarrow \mathtt{inside}_t(\mathtt{A},\mathtt{C}) \sim= \mathtt{true}, \mathtt{tr\_inside}_t(\mathtt{C},\mathtt{B}).\end{aligned}$$

In addition, we assume that the probability of observing an object inside a box is null. Therefore, to improve the performance we consider a proposal distribution for $\mathtt{inside}_t(\mathtt{A},\mathtt{B})$ that depends on the observation of A.

For the packaging scenario we performed several test-cases. We assume the type, size and color are known for each object. We also encoded the static object 'table', therefore when an object is not held it will fall down until it collides with another object or the table. The first test-case consists in taking a box, putting an object inside the box, moving the box and then rotating the box upside down. The second test-case consists in putting an object inside a small box, putting the small box in a bigger box, then moving the bigger box and rotating it upside down. Finally, we put an object inside a small box and then rotate the box on top of another box, the object inside has to fall inside the other object. For this scenario we

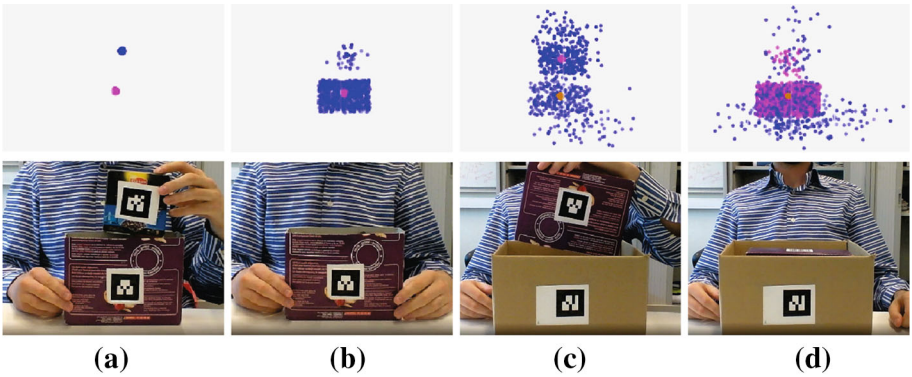

Fig. 12 Packaging scenario experiments. The bottom images represent moments of the experiment, while the top images show the corresponding estimated objects' positions, where each colored point represents an object in a sample. The cube is represented in blue, the small box in fuchsia and the big box in beige. (a) cube on the small box. (b) cube inside the small box (c) rotated small box on the big box. (d) cube and box inside the big box

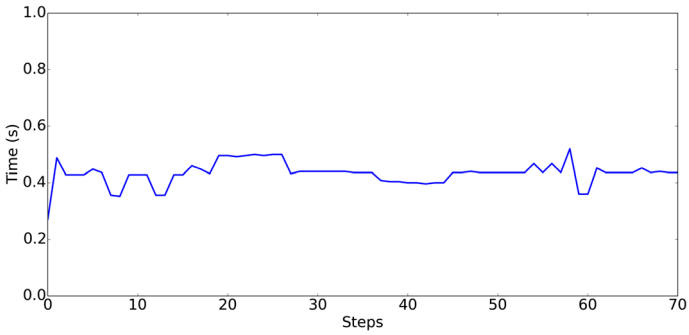

Fig. 13 Inference time per step in the packaging scenario, with 3 objects and 500 samples

used 600 samples. The results (Fig. 12) showed that the model is stable, correctly tracks the objects, and successfully estimates the transitive relation inside (Q4). For example, whenever we put an object in a box the DCPF estimates that it is inside the box or still outside with a small probability. Similarly, when we rotate a box upside down the object that was inside falls outside and goes inside the box below. One issue that was encountered is when the objects are moved rapidly, in this case the filter keeps track of the visible objects but may lose the sample diversity of the invisible objects. To avoid this problem the variance in the state transition and observation model needs to be increased.

In this scenario we can consider queries such as how many objects there are in the box with the respective probability for each object, or if there is a blue cube in the red box. The second query would be the conjunction: $\texttt{type(A, cube)}, \texttt{color(A, blue)}, \texttt{inside}_t\texttt{(A, B)} \sim= \texttt{true}, \texttt{type(B, box)}, \texttt{color(B, red)}$. The answer is the probability that the query is true. Alternatively we can list all object pairs $\texttt{(A, B)}$ that satisfy the query with the respective probability.

The filter performance for this scenario depends on the number of objects, the framework spends around 0.37 *ms*, 0.6 *ms*, 0.87 *ms* and 1.08 *ms* per sample respectively for 1, 2, 3 and 4 objects, assuming all objects are visible. If some objects are not visible the performance is better. Figure 13 shows an example of execution time per step with 3 objects and 500 samples.

Fig. 14 Learnsize scenario. Sketch on the *left*: the objects are pushed away from each other when they overlap, applying a displacement. Picture on the center with 3 objects. The *right figure* represents the estimated objects' positions (*yellow*, *orange* and *grey*), and the estimated size (one point per sample) using artificial dynamics. The *blue lines* are the real size and the *black lines* the average estimated size. The distance is measured in meters

7.4 Learnsize scenario

To illustrate the described learning algorithms and their adaptation for DCPF we consider an object tracking scenario called Learnsize. In this scenario we have a table with several objects. The goal is to track the objects moved by a human (or a robot), and estimate online the size of the objects from their interaction. This problem involves static parameters, that is, the objects' size. This makes the problem difficult as explained in Sect. 6.1. The **state variables** to estimate are the object positions, the size for each object (i.e., diameter assuming round objects), and the object ID moved by a human or robot (if any). The **observation** is a set of noisy object positions, and there are no actions. Whenever the objects touch (or overlap) each other, each object is pushed away from the other one (Fig. 14). The actual objects never overlap, nonetheless this can happen in the samples. The overlap occurs when the distance between the center of the two objects is smaller than the sum of the objects' radiuses (average of the diameter). If there is an overlap we apply a displacement proportional to the absolute difference between the distance and the radiuses sum. Multiple displacements can be applied to the same object, in that case the total displacement is the sum of the simple components. However, whenever the object is held we assume the displacement is 0. The object size distribution is defined in Sect. 6.2 according to the learning method.

In this scenario we need to estimate which object is held and moved by a human. To simplify the problem we assume the human can move at most one object at a time. Thus, we added the variable $\texttt{move}_t$ in the state that indicates the object ID held/moved or zero for none. We defined the following transition model, which considers more probable to remain in the same state:

$$\begin{aligned}\texttt{move}_{t+1} \sim \texttt{uniform}([\simeq(\texttt{move}_t), \simeq(\texttt{move}_t)|\texttt{L}]) \leftarrow \\ \texttt{findall}(\texttt{ID}, \texttt{object}(\texttt{ID}) \sim= \texttt{V}, \texttt{L}).\end{aligned}$$

Whenever $\texttt{move}_t$ has the value ID the transition model for object ID will have a noise variance double that of the other unmoved objects, e.g., for the axis x:

$$\begin{aligned}\texttt{pos}_\texttt{x}(\texttt{ID})_{t+1} &\sim \texttt{gaussian}(\simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_t), \sigma^2) \leftarrow \simeq(\texttt{move}_t) = \texttt{ID}. \\ \texttt{pos}_\texttt{x}(\texttt{ID})_{t+1} &\sim \texttt{gaussian}\left(\simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_t) + \simeq(\texttt{totDisp}_\texttt{x}(\texttt{ID})_t), \frac{\sigma^2}{2} \right) \leftarrow \\ &\quad \simeq(\texttt{move}_t) \neq \texttt{ID}. \end{aligned} \tag{29}$$

Table 1 Learnsize scenario results

Algorithm	Correct	Avg error (cm)	Time per sample (ms)
Artificial dynamics	27/30	0.7	1.6
Storvik's filter variation	23/30	1.3	2.4

Avg error is the absolute distance between the ground truth and the averaged estimation of the objects size (averaged over over objects and trials). 'Correct' is the total number of objects size estimated correctly, that is with an error below 1.5 cm

If the object is not held we take in account the eventual displacements caused collisions with other objects. Thus, $\texttt{totDisp}_{\texttt{x}}(\texttt{ID})_{\texttt{t}}$ is the sum of all displacements $\texttt{displacement}_{\texttt{x}}(\texttt{ID}, \texttt{C})_{\texttt{t}}$ applied to object ID along the x axis and caused by contact with object C. Finally, to improve the performance we used a suboptimal proposal distributions (see "Proposal distribution" in the Appendix for details).

We tested the two described learning algorithms in the Learnsize scenario: artificial dynamics and the proposed Storvik's filter variation. We performed 10 trials for each of the two algorithms for three objects. In each trial we randomly pulled and pushed one object at a time. The results of the experiments are summarized in Table 1 and were performed using 700 samples. In the experiments with three objects both learning strategies perform reasonably well (Q5). Artificial dynamics performs better and is faster than Storvik's filter variation. An example with three objects using artificial dynamics is shown in Fig. 14. Learning becomes harder with a higher number of objects. For better performance offline methods are required. In addition, the performance are highly dependent on the pushes/pulls performed. Indeed, there are different objects sizes that can produce a similar behavior when moved. To make the algorithms converge to the ground truth all objects were put in contact with all the others during the experiments[5]. The videos of the described and other experiments are available at https://dtai.cs.kuleuven.be/ml/systems/DC/.

8 Related work

8.1 Frameworks

In this section we will review related frameworks for static and then dynamic inference.

The proposed static inference is related to BLOG (Milch et al. 2005) inference and to Monte-Carlo inference used in ProbLog (Kimmig et al. 2008). As discussed in Sect. 7.2 BLOG is based on LW and lazy instantiation as EvalSampleQuery. However, BLOG exploits LW only for simple evidence statements, thus it performs worse than DC with complex queries described in Sect. 3.3.2. Furthermore, many probabilistic languages [e.g., Anglican (Wood et al. 2014), Church (Goodman et al. 2008) and BLOG] are not always able to give an answer for complex queries with evidence containing continuous variables as shown in the experiments (for BLOG). In contrast, DC gives meaningful answers in those cases exploiting LW in a larger set of cases.

For dynamic inference, DCPF is related to probabilistic programming languages such as BLOG, Church (Goodman et al. 2008), ProbLog (Kimmig et al. 2008), and the Distribu-

[5] This is valid for 3 or more objects, with 2 objects it is not possible to learn the objects size from pushes. Indeed, any pair of objects with the same sum of sizes produces the same behaviour, because the distance between the objects during contact is the same.

tional Clauses of Gutmann et al. (2011). While these languages are expressive enough to be used for modeling dynamic relational domains, these languages do not support explicitly filtering (BLOG excluded), which makes inference prohibitively slow or unreliable for dynamic models. Also worth mentioning is first-order logical filtering (e.g., see Shirazi and Amir 2011) the logical deterministic counterpart of probabilistic filtering. This method can inspire further DCPF extensions, nonetheless the absence of a probabilistic framework and continuous distributions make them less suitable for the range of applications considered in this paper.

There exist probabilistic programming approaches for temporal models. A variant of BLOG for filtering in dynamic domains (de Salvo Braz et al. 2008) has been proposed, it instantiates the variables needed for inference as BLOG. However, as discussed in Sect. 7.2, DBLOG does not currently implement Step (2) of our filtering algorithm. This requires one to manually query all the variables that d-separate from the past or at least those that might be relevant for the given query. In contrast, DCPF automatically determines which variable to samples to guarantee d-separation, exploiting context specific independences. This avoids backinstantiation with the possibility to use lifted beliefs update and precomputed beliefs. Furthermore, all the considerations about LW in complex queries and continuous evidence are valid for the dynamic case.

Logical HMMs (Kersting et al. 2006) employ logical atoms as observations and states and hence, their expressivity is more limited. The lifted relational Kalman filter (Choi et al. 2011), performs efficient lifted exact inference for continuous dynamic domains, but it assumes linear Gaussian models. The relational particle filter of Manfredotti et al. (2010) cannot handle partial samples. Finally, the approaches that are most similar to ours are those of Zettlemoyer et al. (2007) and Probabilistic Relational Action Model (PRAM) (Hajishirzi and Amir 2008). The former employs first-order formulas to represent a set of states called hypothesis; these are similar to our partial worlds in that they represent a potentially infinite number of states. The key difference is that our approach explicitly defines random variables, (in)dependence assumptions, and their conditional distributions in relationship to other random variables, which allows us to efficiently compute the distribution of a random variable that needs to be sampled and added to the sample. In PRAM the filtering problem is converted into a deterministic first-order logic problem that can be solved using progression, regression and sampling. PRAM is mainly suited for relational domains, that are inherently discrete and binary. In addition, PRAM performs regression of a formula from time t to time 0 that implies performance issues as previously discussed.

Furthermore, none of the frameworks of Thon et al. (2011), Kersting et al. (2006), Zettlemoyer et al. (2007), Hajishirzi and Amir (2008), Natarajan et al. (2008) supports continuous random variables (other than through discretization), therefore these techniques cannot deal with real-world applications in robotics. Discretization is not always a good solution, and it can dramatically increase the number of states, therefore it is unclear whether these algorithms would maintain good performance in such cases. Finally, their first-order logic representation allows discrete and fixed probabilities (for the transition and measurement model), instead DCPF provides a flexible language to represent continuous and discrete distributions that can be parameterized by other random variables or logical variables used in the body. This allows a compact model and faster inference.

Several improvements to classical particle filtering have been proposed, such as Rao–Blackwellization (Casella and Robert 1996) and Factored Particle Filtering (Pfeffer et al. 2009). The first method has been exploited in our approach, but further improvements are possible. Factored Particle Filters cluster the state space reducing the variance and improving

the accuracy. These methods are complementary to our work and could be adapted in future work.

8.2 Applications

Some state estimation applications with a relational representation have been proposed. The relational particle filter of Cattelani et al. (2012) uses relations such as 'walking together' in people tracking to improve prediction and the tracking process. They divide the state in two sets: object attributes and relations, making some assumptions to speed up inference. In their approach a relation can be true or false. In contrast, our approach does not make a real distinction between attributes and relations, indeed, each random variable has a relational representation, regardless of the distribution (binary, discrete or continuous). This allows parametrization and template definition for any kind of random variable. Furthermore, our language and inference algorithm are more general, keeping inference relatively fast. In addition, it is not clear if they can support partial states and integrate background knowledge while keeping good performance. A relational representation has been used in Meyer-Delius et al. (2008) for situation characterization over time. However, this work is based on HMMs and uses only binary relations (true or false). Interesting works have been proposed (Beetz et al.; Tenorth and Beetz 2009) for manipulation tasks exploiting a relational representation. Those works integrate relational knowledge about the world to reason about the objects and perform complex tasks. For probabilistic inference and belief update they use MLNs (Markov Logic Networks). However, MLNs are arguably less efficient for filtering inference because they are undirected models that might require MCMC or the computation of the partition function at each step, even though recent optimizations have been proposed (Papai et al. 2012).

9 Conclusions

We proposed a flexible representation for hybrid relational domains and provided an efficient inference algorithm for static and dynamic models. This framework exploits the relational representation and (context specific) independence assumptions to reduce the sample size (through partial worlds) and the inference cost.

The proposed static algorithm EvalSampleQuery exploits LW in a wider range of cases with respect to systems such as BLOG, and supports complex queries with continuous variables, for which most related frameworks fail. These features are also valid for dynamic domains, where DCPF calls EvalSampleQuery during filtering. At the same time, DCPF avoids backinstantiation to bound the space complexity and reduce time performance variability. This makes DCPF particularly suited for online applications.

One of the advantages of a relational framework like DCPF is the flexibility and generality of the model with respect to a particular situation. Indeed, whenever a new object appears all the respective properties and relations with other objects are implicitly defined (but not necessary computed and added to the sample). In addition, the expressivity of the language helps to bridge the gap between robotics and the high-level symbolic representation used in Artificial Intelligence.

The static algorithm EvalSampleQuery and the DCPF filtering were empirically evaluated and applied in several synthetic and real-world scenarios. The results show that EvalSampleQuery outperforms naive MC and BLOG in static domains. For dynamic domains DCPF outperforms the classical particle filter and DBLOG for a small number of

samples. The DCPF averaged error is lower than the DBLOG error for the same number of samples. Nonetheless, DBLOG seems to be faster for a large number of samples, which might be caused by implementation reasons.

The object tracking experiments show that DCPF is promising for robotics applications. The overall performance is acceptable, but could be improved to scale well with high-dimensional states. Indeed, each sample represents the entire state, therefore inference can be computationally intensive for a high number of objects and relations. Nonetheless, DCPF exploits the structure of the model and partial samples to speed up inference and improve the performance.

Finally, both learning strategies tested in this framework perform reasonably well for a limited number of parameters. More sophisticated strategies and offline methods need to be investigated for a higher number of parameters.

Appendix

Logic programming

In this appendix we briefly introduce logic programming concepts. See Nilsson and Małiszyński (1995), Apt (1997), Lloyd (1987) for an extensive introduction.

An atomic formula (atom) is a predicate applied to a list of terms that represents objects. For example, $\texttt{inside}(1, 2)$ is an atomic formula, where $\texttt{inside}$ is a predicate, sometimes called relation, and 1, 2 are symbols that refer to objects. A literal is an atomic formula or a negated atomic formula. A *clause*, in logic programming, is a first-order formula with a head (atom), and a body (a list of literals). For example, the clause

$$\texttt{inside}(\texttt{A}, \texttt{B}) \leftarrow \texttt{inside}(\texttt{A}, \texttt{C}), \texttt{inside}(\texttt{C}, \texttt{B})$$

states that for all $\texttt{A}$, $\texttt{B}$ and $\texttt{C}$, $\texttt{A}$ is inside $\texttt{B}$ if $\texttt{A}$ is inside $\texttt{C}$ and $\texttt{C}$ is inside $\texttt{B}$ (transitivity property). $\texttt{A}$, $\texttt{B}$ and $\texttt{C}$ are logical variables, that informally refer to an arbitrary object. A clause usually contains non-ground literals, that is, literals with logical variables (e.g., $\texttt{inside}(\texttt{A}, \texttt{B})$). A clause with logical variables is assumed to be preceded by universal quantifiers for each logical variable, e.g., in the above clause: $\forall\texttt{A}$, $\forall\texttt{B}$, $\forall\texttt{C}$. A substitution θ replaces the variables with other terms (eventually other variables). For example, for $\theta = \{\texttt{A} = 1, \texttt{B} = 2, \texttt{C} = 3\}$ the above clause becomes:

$$\texttt{inside}(1, 2) \leftarrow \texttt{inside}(1, 3), \texttt{inside}(3, 1)$$

and states that if $\texttt{inside}(1, 3)$ and $\texttt{inside}(3, 1)$ are true, then $\texttt{inside}(1, 2)$ is true. We indicate with $\theta = \texttt{mgu}(\texttt{A}, \texttt{B})$ the most general unifier, i.e. the most general substitution θ that makes $\texttt{A}\theta = \texttt{B}\theta$.

Let $\mathbb{P}$ be a definite program and I a Herbrand interpretation. The $T_P(I)$ operator is defined as follows:

$$T_P(I) = \{h\theta | h \leftarrow b_1, \ldots, b_n \in \mathbb{P}, \{b_1\theta, \ldots, b_n\theta\} \subseteq I\}$$

That is if the body of a rule is true in I, the head is in $T_P(I)$. Given a program it is possible to derive all possible true atoms using the T_P operator a number of times recursively starting from $I = \emptyset$, until a fixpoint is reached (i.e., $T_P(I) = I$). The interpretation obtained is called Least Herbrand Model and contains all the atoms that can be derived from $\mathbb{P}$.

Distributional clauses

Validity conditions

To define a proper probability distribution $p(x)$, a DC program $\mathbb{P}$ needs to satisfy the validity conditions described in Gutmann et al. (2011). For each predicate h or random variable we assign a rank (natural number) that defines an order. A DC program is valid if:

1. For each ground $\mathtt{h}\theta$, $\mathtt{h}\theta \sim \mathcal{D}\theta$ has to be unique in the least fixpoint, i.e., there is one distribution defined for each random variable.
2. The program has to be stratified, that is, there exists a rank assignment such that for each distributional clause $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n. : rank(\mathtt{h} \sim \mathcal{D}) > rank(\mathtt{b}_i)$, while each definite clause $\mathtt{h} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n. : rank(\mathtt{h}) \geq rank(\mathtt{b}_i)$.
3. All ground probabilistic facts are Lebesgue-measurable.
4. Each atom in the least fixpoint can be derived from a finite number of probabilistic facts (finite support condition Sato 1995).

ST_P operator

To generate a possible world, Gutmann et al. (2011) define the ST_P operator, a stochastic version of the well-known T_P operator in logic programming. This operator is applied on partial worlds (or interpretations); these contain ground atoms (as in standard logic programming), and for each random variable $\mathtt{r}$ defined in the partial worlds, there will be an atom of the form $\mathtt{r} \sim \mathcal{D}$, and an equality $\mathtt{r} = v$ in a separated table, where v is the value sampled from the distribution $\mathcal{D}$. Throughout the paper we do not always write the $\mathtt{r} \sim \mathcal{D}$ explicitly. To generate a possible world, one starts from the empty partial world $I = \emptyset$, and applies $I \leftarrow ST_P(I)$ until a fixpoint is reached ($ST_P(I) = I$).

Negation

A distributional program $\mathbb{P}$ needs to be stratified to be valid, thus no specific requirements are needed to support negation. The semantics is defined along the same lines as the perfect model M_P (Przymusinski 1988), that is, the ST_P operator is applied at each rank from lowest to highest rank. The result is a world x sampled from the distribution $p(x)$ defined by the program $\mathbb{P}$.

Consider a DC program $\mathbb{P}$, a literal l and a (partial) world $x^{P(i)}$ obtained by applying the ST_P operator till the least fixpoint for $rank(var(l))$ is reached (or exceeded). If l is a classic atomic formula, l is true in $x^{P(i)}$ iff $l \in x^{P(i)}$, and $not(l)$ holds in $x^{P(i)}$ iff $l \notin x^{P(i)}$. This follows the closed world assumption: a literal is false if it is not in the least fixpoint. If l is a comparison operator involving a random variable r: $l = (r \sim\!= val)$, then $not(r \sim\!= val)$ holds in $x^{P(i)}$ whenever $r \sim\!= val$ is false in $x^{P(i)}$ or the random variable r is not defined in $x^{P(i)}$, that is, when $r \notin var(x^{P(i)})$. Note that $x^{P(i)}$ is the least fixpoint for $rank(r)$ (or higher), thus $r \notin var(x^{P(i)})$ implies that r is not defined for each world $m \supseteq x^{P(i)}$ consistent with $x^{P(i)}$. To determine $not(l)$, $not(r \sim\!= val)$ or just the existence of a variable r it is not required to explicitly apply the ST_P operator. In logic programming, negation as failure is used to prove negated formulas: if the query q fails then $not(q)$ is true and vice versa. A common inference procedure is SLDNF, that is SLD resolution with negation as failure.

Consider the following examples:

$$\mathtt{n} \sim \mathtt{poisson(6)}. \tag{30}$$

$$\texttt{color(X)} \sim \texttt{uniform([red, blue, black])} \leftarrow \texttt{n} \sim= \texttt{N}, \texttt{between(1, N, X)}. \quad (31)$$

$$\texttt{notred} \leftarrow \texttt{not(color(2)} \sim= \texttt{red)}. \quad (32)$$

$$\texttt{nothing\_red} \leftarrow \texttt{not(color(X)} \sim= \texttt{red)}. \quad (33)$$

`notred` is true in those worlds where the value of the random variable `color(2)` is not red. In some worlds, the variable `color(2)` is not defined, for example when $n = 1$, in such cases `notred` still succeeds. Similarly, the atom `nothing_red` is true in a world x iff the query `color(X)` $\sim=$ `red` fails, that is, iff every variable `color(X)` defined in x is not red (for `X` between 1 and n). Therefore, `nothing_red` is also true in worlds with no objects (`n` $= 0$).

Theorems

A distributional program describes a conditional distribution, and independence assumptions as described in the following theorem.

Theorem 4 *Given a valid DC program $\mathbb{P}$, a DC clause $\texttt{r} \sim \mathcal{D} \leftarrow \texttt{body}$ and a partial world $x^{P(i)}$, if there exists a grounding substitution θ such that $(\forall v : v \in var(x^{P(i)}) \Rightarrow rank(v) \leq rank(\texttt{r}\theta))$ and $(x^{P(i)} \models \texttt{body}\theta)$ then $p(r\theta|x^{P(i)}) = p(\texttt{r}\theta|\texttt{body}\theta) = \mathcal{D}\theta$.*

Proof (sketch) The proof can be obtained from the semantics of DC (see Gutmann et al. 2011, or Milch 2006 for a general discussion). The result is similar to Bayesian networks for which each random variable is conditionally independent of its non-descendants given its parents. The same result is valid for context-specific independencies. □

Theorem 5 *Step (1) guarantees d-separation conditions 1 and 2*

Proof We need to prove $p(\hat{x}_{t+1}^{P(i)}|\hat{x}_t^{P(i)}, \hat{x}_t^a, u_{t+1}) = p(\hat{x}_{t+1}^{P(i)}|\hat{x}_t^{P(i)}, u_{t+1})$ and $p(z_{t+1}|x_{t+1}^{(i)}) = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$. The sampling algorithm can sample a random variable r_{t+1} only when the body of a clause that defines r_{t+1} is true in the partial sample, i.e., when there exists a substitution θ and a clause $\texttt{h}_{t+1} \sim \mathcal{D} \leftarrow \texttt{body}_{\texttt{t:t+1}} \in \mathbb{P}$ such that $r_{t+1} = \texttt{h}_{t+1}\theta$, $var(\texttt{body}_{\texttt{t:t+1}}\theta) \subseteq var(\hat{x}_{t:t+1}^{P(i)})$ and $\hat{x}_{t:t+1}^{P(i)} \models \texttt{body}_{\texttt{t:t+1}}\theta$. From Theorem 4, $p(r_{t+1}|\texttt{body}_{\texttt{t:t+1}}\theta, \hat{x}_t^{P(i)}, \hat{x}_t^a, u_{t+1}) = p(r_{t+1}|\texttt{body}_{\texttt{t:t+1}}\theta)$ holds for each $r_{t+1} \in \hat{x}_{t+1}^{P(i)}$, this proves condition 1 since $var(\texttt{body}_{\texttt{t:t+1}}\theta) \subseteq var(\hat{x}_{t:t+1}^{P(i)})$. Similarly, the sampling algorithm will sample variables that prove the evidence z_{t+1}; by applying Theorem 4 again we have $p(z_{t+1}|x_{t+1}^{(i)}) = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$. □

Theorem 6 *Step (2) guarantees d-separation condition 3.*

Proof Condition 3 is satisfied iff the posterior distribution of the marginalized variables: $f_{t+1}^{(i)}(\hat{x}_{t+1}^a; \hat{x}_{t+1}^{P(i)}) = \prod_{r_{t+1} \in \hat{x}_{t+1}^a} f_{t+1}^{(i)}(\texttt{r}_{\texttt{t+1}}; Parents(\texttt{r}_{\texttt{t+1}}))$ is computed for each sample. The probability distribution $f_{t+1}^{(i)}$ is represented using DDC clauses or storing $\texttt{r}_{\texttt{t+1}} \sim \mathcal{D}$ in the sample. There are two cases to discuss for each random variable $\texttt{r}_{\texttt{t+1}} \in \hat{x}_{t+1}^a$:

c1 $\texttt{r}_{\texttt{t+1}}$ is defined with a grounded DC clause of the form $\texttt{h}_{\texttt{t+1}}\theta \sim \mathcal{D}\theta \leftarrow \texttt{body}_{\texttt{t+1}}\theta$ (derived from $\texttt{h}_\texttt{t} \sim \mathcal{D} \leftarrow \texttt{body}_\texttt{t}$), with $\texttt{r}_{\texttt{t+1}} = \texttt{h}_{\texttt{t+1}}\theta$. In this case no actions are required because in the next step $\texttt{r}_{\texttt{t+1}} \rightarrow \texttt{r}_\texttt{t}$ and $f_t^{(i)}(\texttt{r}_\texttt{t}; Parents(\texttt{r}_\texttt{t}))$ is defined by $\texttt{h}_\texttt{t}\theta \sim \mathcal{D}\theta \leftarrow \texttt{body}_\texttt{t}\theta$. The variable $\texttt{r}_{\texttt{t+1}}$ may depend on random variables $\texttt{d}_{\texttt{t+1}} \in \texttt{body}_{\texttt{t+1}}\theta$ that are not yet sampled: $\texttt{d}_{\texttt{t+1}} \in \hat{x}_{t+1}^a$. In this case, if $\texttt{d}_{\texttt{t+1}}$ is defined by intra-time clauses of the form $\texttt{d}_{\texttt{t+1}} \sim \mathcal{D} \leftarrow \texttt{dbody}_{\texttt{t+1}}$, the case c1 applies recursively. If $\texttt{d}_{\texttt{t+1}}$ is defined by inter-time clauses: $\texttt{d}_{\texttt{t+1}} \sim \mathcal{D} \leftarrow \texttt{dbody}_{\texttt{t:t+1}}$, the case c2 applies.

c2 $\mathtt{r_{t+1}}$ is defined with a grounded DC clause of the form $\mathtt{h_{t+1}}\theta \sim \mathcal{D}\theta \leftarrow \mathtt{body_{t:t+1}}\theta$ (derived from the state transition model) with $\mathtt{r_{t+1}} = \mathtt{h_{t+1}}\theta$. In this case Step (2) queries $\mathtt{r_{t+1}}$, thus variables in $\mathtt{body_{t:t+1}}\theta$ will be eventually sampled if not in $\hat{x}^{P(i)}_{t:t+1}$. If $\mathtt{body_{t:t+1}}\theta$ is true in $\hat{x}^{P(i)}_{t:t+1}$, Step (2) adds $\mathtt{r_{t+1}} \sim \mathcal{D}\theta$ is to the sample. Indeed, from Theorem 2 we have $f^{(i)}_{t+1}(\mathtt{r_{t+1}}; Parents(\mathtt{r_{t+1}})) = p(\mathtt{r_{t+1}}|\hat{x}^{P(i)}_t, \hat{x}^{P(i)}_{t+1}) = \mathcal{D}\theta$.

□

The case c1 implies that Step (2) does not need to query variables defined by intra-time clauses $\mathtt{h_{t+1}} \sim \mathcal{D} \leftarrow \mathtt{body_{t+1}}$. Querying those variables would sample unnecessary random variables. For the remaining variables, Step (2) performs the prediction step, that is, determines the distribution of such variables (case c2). If lifted belief update or precomputed belief are used for a random variable r_{t+1}, the marginal distribution $f^{(i)}_{t+1}$ of such variable is defined by DDC clauses. In conclusion, c1 and c2 show that Step (2) guarantees that the distributions of the marginalized variables are defined in any situation.

Proposal distribution

To make the particle filter more efficient the optimal proposal distribution $p(x_{t+1}|x_t, z_{t+1})$ and the corresponding weight $p(z_{t+1}|x_t)$ can be used. Given a complex nonlinear transition model those distributions are not easy to compute analytically, therefore we adopt suboptimal solutions. Let us assume that the state is $x_t = \{a_t, b_t\}$ and the observations depend only on b_{t+1}, then the weight can be written as:

$$w^{(i)}_{t+1} = w^{(i)}_t \frac{p(z_{t+1}|b^{(i)}_{t+1}) p(b^{(i)}_{t+1}|a^{(i)}_{t+1}, x^{(i)}_t) p(a^{(i)}_{t+1}|x^{(i)}_t)}{g(x^{(i)}_{t+1}|x^{(i)}_t, z_{t+1})}$$

A suboptimal proposal distribution is $g(x_{t+1}|x^{(i)}_t, z_{t+1}) = p(b_{t+1}|a_{t+1}, x^{(i)}_t, z_{t+1})$ $p(a_{t+1}|x^{(i)}_t)$, with weight $w^{(i)}_{t+1} = w^{(i)}_t p(z_{t+1}|a^{(i)}_{t+1}, x^{(i)}_t)$. If $p(b_{t+1}|a_{t+1}, x_t)$ is a linear Gaussian transition model or discrete we can easily compute the above suboptimal proposal and relative weight after sampling a_{t+1}. In the scenarios, b_t is the set of the objects' positions, while the remaining states define the set a_t. For example, the proposal that replaces (29) is:

$$\begin{aligned} \mathtt{pos_x(ID)_{t+1}} \sim \mathtt{gaussian(M, Var)} \leftarrow \mathtt{Var\ is}\ (\sigma^{-2} + \omega^{-2})^{-1}, \\ \mathtt{M\ is\ Var} * (\simeq(\mathtt{obsPos_x(ID)_t})\omega^{-2} + \simeq(\mathtt{pos_x(ID)_t})\sigma^{-2}), \\ \simeq(\mathtt{move_t}) = \mathtt{ID}. \end{aligned} \tag{34}$$

Where ω^2 is the variance of the Gaussian measurement model.

Comparison with Murphy's interface algorithm

Murphy (2002) proposed the interface algorithm for Dynamic Bayesian Networks to perform efficient exact filtering. It is based on the notion of interface: the set of variables that have children in the next time slide.

In DCPF, we can define the interface as the set of random variables that appears in the body of a clause of the state transition model. The interface is sufficient to d-separate the future from the past. Thus, Step (2) can perform the prediction step only for random variables in the interface. However, to query a non-interface variable h_t, the partial sample $x^{P(i)}_{t-1}$ is required in addition to $x^{P(i)}_t$ (while $x^{P(i)}_{0:t-2}$ can be forgotten). This is because the prediction

step is not performed for non-interface variables. Unfortunately, the interface is fixed and does not consider context-specific independencies, therefore the non-interface set might be empty or small in several domains.

For the described reasons the interface concept is less appealing for inference optimization in DCPF. Therefore, the interface is not exploited in the current implementation. Nonetheless, some domains might benefit of this improvement.

References

Andrieu, C., Doucet, A., & Tadic, V. B. (2005). On-line parameter estimation in general state-space models. In *Proceedings of the 44th IEEE conference on decision and control, 2005 and 2005 European control conference (CDC-ECC '05)*, pp. 332–337.

Apt, K. (1997). *From logic programming to Prolog*. Upper Saddle River: Prentice-Hall international series in computer science. Prentice Hall.

Bancilhon, F., & Ramakrishnan, R. (1986). An amateur's introduction to recursive query processing strategies. *SIGMOD Record*, *15*, 16–51.

Beetz, M., Jain, D., Mosenlechner, L., Tenorth, M., Kunze, L., & Blodow, N., et al. (2012) Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, *100*(8), 2454–2471.

Carvalho, C. M., Johannes, M. S., Lopes, H. F., & Polson, N. G. (2010). Particle learning and smoothing. *Statistical Science*, *25*(1), 88–106.

Carvalho, C. M., Lopes, H. F., Polson, N. G., & Taddy, M. A. (2010). Particle learning for general mixtures. *Bayesian Analysis*, *5*(4), 709–740.

Casella, G., & Robert, C. P. (1996). Rao-Blackwellisation of sampling schemes. *Biometrika*, *83*(1), 81–94.

Cattelani, L., Manfredotti, C., & Messina, E. (2012). A particle filtering approach for tracking an unknown number of objects with dynamic relations. *Journal of Mathematical Modelling and Algorithms in Operations Research*, *13*(1), 3–21.

Choi, J., Guzman-Rivera, A., & Amir, E. (2011). Lifted relational Kalman filtering. In *Proceedings of the 22nd international joint conference on artificial intelligence (IJCAI 2011)*, pp. 2092–2099.

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (Eds.). (2008). *Probabilistic inductive logic programming, theory and applications. Lecture notes in computer science*, Vol. 4911. Berlin: Springer.

de Salvo Braz, R., Arora, N., Sudderth, E., & Russell, S. (2008). Open-universe state estimation with DBLOG. In *NIPS workshop on probabilistic programming: Universal languages, systems and applications*.

Doucet, A., de Freitas, N., Murphy, K., & Russell, S. (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the 16th conference on uncertainty in artificial intelligence (UAI '00)* (pp. 176–183). Burlington: Morgan Kaufmann.

Doucet, A., Godsill, S., & Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, *10*(3), 197–208.

Fung, R. M., & Chang, K. (1989). Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Proceedings of the 5th conference on uncertainty in artificial intelligence (UAI 1989)*.

Getoor, L., & Taskar, B. (2007). *An introduction to statistical relational learning*. Cambridge, MA: MIT Press.

Gilks, W. R., & Berzuini, C. (2001). Following a moving targetmonte carlo inference for dynamic bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *63*(1), 127–146.

Goodman, N., Mansinghka, V. K., Roy, D. M., Bonawitz, K., & Tenenbaum, J. B. (2008). Church: A language for generative models. In *Proceedings of the 24th conference on uncertainty in artificial intelligence (UAI 2008)* (pp. 220–229). Edinburgh: AUAI Press.

Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., & De Raedt, L. (2011). The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming, 11*, 663–680.

Hajishirzi, H., & Amir, E. (2008). Sampling first order logical particles. In *Proceedings of the 24th conference on uncertainty in artificial intelligence (UAI 2008)* (pp. 248–255). Edinburgh: AUAI Press.

Higuchi, T. (2001). Self-organizing time series model. In A. Doucet, N. Freitas, & N. Gordon (Eds.), *Sequential Monte Carlo methods in practice, statistics for engineering and information science* (pp. 429–444). New York: Springer.

Kadane, J. (2011). *Principles of uncertainty*. Abingdon: Taylor & Francis.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, *82*, 35–45.

Kantas, N., Doucet, A., Singh, S. S., & Maciejowski, J. M. (2009). An overview of sequential monte carlo methods for parameter estimation in general state-space models. In *15th IFAC symposium on system identification*, Vol. 15, (pp. 774–785).

Kersting, K., De Raedt, L., & Raiko, T. (2006). Logical hidden Markov models. *Journal of Artificial Intelligence Research*, *25*, 425–456.

Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., & De Raedt, L. (2008). On the efficient execution of ProbLog programs. In *Proceedings of the 24th conference on logic programming (ICLP 2008). Lecture notes in computer science* (Vol. 5366, pp. 175–189). Berlin: Springer.

Kitagawa, G. (1996). Monte Carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, *5*(1), 1–25.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: Principles and techniques—adaptive computation and machine learning*. Cambridge, MA: MIT Press.

Lemieux, C. (2009). *Monte Carlo and Quasi-Monte Carlo sampling* (Vol. 20). Berlin: Springer.

Lloyd, J. (1987). *Foundations of logic programming*. New York: Springer.

Lloyd, J., & Shepherdson, J. (1991). Partial evaluation in logic programming. *The Journal of Logic Programming*, *11*(34), 217–242.

Lopes, H. F., Carvalho, C. M., Johannes, M., & Polson, N. G. (2010). Particle learning for sequential bayesian computation. In J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, A. F. M. Smith, M. West (Eds.), *Bayesian statistics 9* (Vol. 9, pp. 317–360). Oxford: Oxford University Press.

Manfredotti, C. E., Fleet, D. J., Hamilton, H. J., & Zilles, S. (2010). Relational particle filtering. *NIPS Workshop on Monte Carlo methods for modern applications*, December 2010.

Meyer-Delius, D., Plagemann, C., Wichert, G., Feiten, W., Lawitzky, G., & Burgard, W. (2008). A probabilistic relational model for characterizing situations in dynamic multi-agent systems. In *Data analysis, machine learning and applications*. Berlin: Springer.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., & Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In *Proceedings of the 19th international joint conference on artificial intelligence (IJCAI 2005)*, pp. 1352–1359.

Milch, B., Marthi, B., Sontag, D., Russell, S., Ong, D. L., & Kolobov, A. (2005). Approximate inference for infinite contingent Bayesian networks. In R. G. Cowell, Z. Ghahramani (Eds.), *Proceedings of the 10th international workshop on artificial intelligence and statistics (AISTATS 2005)* (pp. 238–245). Society for Artificial Intelligence and Statistics.

Milch, B. C. (2006). *Probabilistic models with unknown objects*. Ph.D. thesis, University of California, Berkeley.

Murphy, K. P. (2002). *Dynamic Bayesian networks: Representation, inference and learning*. Ph.D. thesis, University of California, Berkeley.

Natarajan, S., Bui, H. H., Tadepalli, P., Kersting, K., & Keen Wong, W. (2008). Logical hierarchical hidden markov models for modeling user activities. In *Proceedings of the 18th international conference in inductive logic programming (ILP 2008). Lecture notes in computer science* (Vol. 5194, pp. 192–209).

Ng, B., Peshkin, L., & Pfeffer, A. (2002). Factored particles for scalable monitoring. In *Proceedings of the 18th conference on uncertainty in artificial intelligence (UAI2002)* (pp. 370–377). Burlington: Morgan Kaufmann.

Nilsson, U., & Małiszyński, J. (1995). *Logic, programming and prolog* (2nd ed.). New York: Wiley.

Nitti, D., Chliveros, G., De Raedt, L., Pateraki, M., Hourdakis, M., & Trahanias, P. (2014a). Application of dynamic distributional clauses for multi-hypothesis initialization in model-based object tracking. In *9th International conference on computer vision theory and applications (VISAPP 2014)*, Vol. 2.

Nitti, D., De Laet, T., & De Raedt, L. (2013). A particle filter for hybrid relational domains. In *Proceedings of the international conference on intelligent robots and systems (IROS 2013)*, pp. 2764–2771.

Nitti, D., De Laet, T., & De Raedt, L. (2014b). Distributional clauses particle filter. In *Machine learning and knowledge discovery in databases, lecture notes in computer science* (Vol. 8726, pp. 504–507). Berlin: Springer.

Nitti, D., De Laet, T., & De Raedt, L. (2014c). Relational object tracking and learning. In *Proceedings of the International conference on robotics and automation (ICRA 2014)*.

Owen, A. B. (2013). Monte Carlo theory, methods and examples. http://statweb.stanford.edu/~owen/mc/

Papai, T., Kautz, H., & Stefankovic, D. (2012). Slice normalized dynamic Markov logic networks. In *Advances in neural information processing systems (NIPS 2012)* (pp. 1907–1915).

Perov, Y., Paige, B., & Wood, F. *The Indian GPA problem*. Retrieved February 23, 2016, from http://www.robots.ox.ac.uk/~fwood/anglican/examples/viewer/?worksheet=indian-gpa.

Pfeffer, A., Das, S., Lawless, D., & Ng, B. (2009). Factored reasoning for monitoring dynamic team and goal formation. *Information Fusion*, *10*(1), 99–106.

Pitt, M. K., & Shephard, N. (1999). Filtering via simulation: Auxiliary particle filters. *Journal of the American statistical association*, *94*(446), 590–599.

Przymusinski, T. C. (1988). Perfect model semantics. In *Proceedings of the 5th international conference on logic programming and symposium (ICLP/SLP 1988)*, pp. 1081–1096.

Robert, C., & Casella, G. (2004). *Monte Carlo statistical methods. Springer texts in statistics*. New York: Springer.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Englewood Cliffs, NJ: Prentice Hall.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th international conference on logic programming (ICLP 1995)* (pp. 715–729). Cambridge, MA: MIT Press.

Shirazi, A., & Amir, E. (2011). First-order logical filtering. *Artificial Intelligence*, *175*(1), 193–219.

Storvik, G. (2002). Particle filters for state-space models with the presence of unknown static parameters. *IEEE Transactions on Signal Processing*, *50*(2), 281–289.

Tenorth, M., & Beetz, M. (2009). KnowRob—Knowledge processing for autonomous personal robots. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS 2009)*, pp. 4261–4266.

Thon, I., Landwehr, N., & De Raedt, L. (2011). Stochastic relational processes: Efficient inference and applications. *Machine Learning, 82*(2), 239–272.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. Cambridge, MA: MIT Press. ISBN: 0262201623.

Whiteley, N., & Johansen, A. M. (2010). Recent developments in auxiliary particle filtering. Barber, C., & Chiappa, (Eds.), *Inference and learning in dynamic models* (pp. 38, 39–47). Cambridge: Cambridge University Press.

Wood, F., van de Meent, J. W., & Mansinghka, V. (2014). A new approach to probabilistic programming inference. In *Proceedings of the 17th international conference on artificial intelligence and statistics (AISTATS 2014)*, pp. 1024–1032.

Zettlemoyer, L. S., Pasula, H. M., & Kaelbling, L. P. (2007). Logical particle filtering. In *Proceedings of the Dagstuhl seminar on probabilistic, logical, and relational learning*.